Refactoring the Knight Class - William

Code Smells Detected-

1. **Variable Names:** - `directionTurner` could be renamed to something like `directionMultiplier` to clarify its purpose.
   - `activeDirections` could be renamed to `movementDirections` to better reflect its meaning.
2. **Method Length:**
   - The `updateEntityPosition()` method is quite long and can be broken down into smaller, more focused methods to improve readability and maintainability.
3. **Low Cohesion:**
   - The `Knight` class seems to have high coupling with the `Play` class, as it directly interacts with it to check for valid movement positions. This could be improved by introducing an abstraction layer or interface to decouple these classes.
4. **Unused Variables:**
   - The variable `directionTurner` is only used for rendering the knight's image. Consider refactoring the rendering logic to remove the need for this variable.
5. **Code Duplication:** - The logic for updating the knight's position based on movement directions could be extracted into a separate method to avoid duplication.
6. **Data Clumps:**
   - The offsets and dimensions used for initializing the collision box could be grouped into a single data structure to avoid passing multiple parameters to methods.

7. **Unjustified Use of Primitives:**
   - Consider using enums instead of boolean flags for `Directions` to improve readability and type safety.

__**What changed**__
- Variable names are improved for clarity.
- The `updateEntityPosition()` method is broken down into smaller methods for better readability.
- The `Knight` class is decoupled from the `Play` class by encapsulating collision detection logic within the knight class.
- Unused variable `directionTurner` is removed.
- Code duplication is reduced by extracting common logic into separate methods.
- Data clumps are avoided by grouping related parameters into a single data structure (`Position` and `Dimension`).

Game Timer - William

Code Smells Detected-

1. **Variable Names**: Rename variables to improve clarity and readability.
2. **Method Length:** Break down long methods into smaller, more focused methods.
3. **Font Initialization**: Move font initialization to a separate method or constructor.
4. **Documentation:** Add comments and documentation to improve code understanding.
5. **Code Structure:** Organize the code to improve its structure and readability.

__**What Changed**__
In this refactored version:
- Variable names are more descriptive.
- Methods are broken down into smaller, focused methods.
- Font initialization is moved to a separate method.
- Comments and documentation are added for better understanding.
- Code structure is improved for readability and maintainability.

Refactoring the UI Buttons:Win,Defeat,Pause and Menu - Fanyi Luo

1. Bad/Confusing Variable Names: Retained original variable names but introduced constants like SCREEN_WIDTH, SCREEN_HEIGHT, BUTTON_WIDTH, and BUTTON_HEIGHT for clarity and to eliminate magic numbers, improving the readability of what these figures represent.
2. Methods that are Too Long: Extracted image loading to a separate method (loadImages) from the constructor to reduce the method length and increase modularity.
3. Low Cohesion: Increased cohesion by ensuring methods like loadImages and initBounds are focused on a single task – setting up button images and initializing button bounds respectively.
4. High Coupling: Coupling was not significantly altered; the class still heavily relies on the Gamestate and external resources, but internal cohesion improvements reduce the complexity of modifications.
5. Lack of Documentation: Improved inline comments and method documentation to clarify the purpose and functionality of each section of the code.
6. Poorly Structured Code: Enhanced structure by clearly defining tasks in methods, using clear naming conventions, and removing duplicated logic.
7. Dead Code: Removed commented-out code blocks and unused comments that were cluttering up the class, thus cleaning up the codebase.
8. Code Duplication: Reduced duplication by consolidating repeated calculations and simplifying the logic for updating the button's state.
9. Unused or Useless Variables: Ensured all variables introduced or retained are used efficiently within the class without redundancy.
10. Data Clumps: Addressed this by grouping related constants together, which makes modifications in dimensions or positioning easier to manage.

<u>What Changed</u>
Introduced Constants: Constants like SCREEN_WIDTH, SCREEN_HEIGHT, BUTTON_WIDTH, and BUTTON_HEIGHT were added, making the code more maintainable and readable by replacing magic numbers.
Simplified Image Loading: The image loading functionality was encapsulated within the loadImages method, simplifying the constructor and focusing on single responsibilities, enhancing modularity.
Reduced Code Duplication: Calculations for centering the button on the screen were centralized in both initBounds and draw methods, reducing duplication and simplifying adjustments.
Enhanced Error Handling: Error messages were enhanced to provide specific details about what went wrong during image loading, aiding in debugging and maintenance.
Cleaned Redundant Code: Unused variables and code related to an unimplemented notPressOver state were removed, streamlining state management within the class.
Improved Logic Readability: The logic for updating the button's visual state (index) based on the pressOver flag was simplified using a ternary operation, making the update method more concise and clear.
Preserved Documentation: JavaDoc comments were preserved and aligned with the changes to maintain consistency and provide clear documentation for class functionality.