**Programming 2 CWK100% : Summative Marking Scheme:**

Highlights show observed features. Red text indicates the feature was missing. General feedback will also be issued which will summarise and cover common issues and good points.

|  |  |
| --- | --- |
| **Base Mark** | **Features Required** (starting at top, tick off each criteria working down the list) |
| 40% | **All** of the following to Pass (40%):   * Player character near top of the screen with **arrow key** controlled left & right movement * Minimum 2 classes : Player, Attacker * At least one attacker object that moves up the screen * Simple working game (comment out code that causes errors) in Processing * Something clearly happens when a player collides with an attacker |
| 40-50% | **All** of the above, and some of the following:   * Player must be restricted to being on the screen * At least 3 attacker objects that move up the screen (and possibly left,right) * Working Collision *function* method(s) |
| 50-60% | **All** of the above, and some of the following:   * Splash or game over screen (draw does different things at different times) * An ArrayList (or array) of attacker objects * Animated sequence of images for the attacker objects (appears to climb, fly, etc) |
| 60-70% | **All** of the above, and some of the following:   * Player can drop objects (attacker removed from game on collision) * A 2nd type (class) of attacker objects that must be avoided by the player (collision involves loss of life or game end) * Class-inheritance for different attacker types (perhaps other classes) * File handling – high score(s) saved and read from file |
| 70-80% | **All** of the above, and some of the following   * Array of PImages for animation sequence * Exhibits some polymorphism with the array/arrayList of attackers |
| 80%+ | **All** of the above, and some of the following:   * collision animation sequence (e.g. explosion) * Complex attacker movements (e.g. sometimes follow the player) * Polymorphism for most (or all) game entities * Use of an Interface or abstract class * Refactored, maintainable code |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **The following table will be completed during marking, where highlighted concepts increased your grade and red concepts decreased your grade within the range indicated in the table above.** | | | | |
| Onscreen:  Score  lives | Splash screen,  Game Over Screen | Game levels | Animated sequence of images: | Classes:  well structured  Appropriate number |
| Meaningful Variable names:  Mostly  All | Constructor(s) | Suitable Methods | Multiple animation sequences | Public/private |
| Meaningful method names,  Parameter names | constants - appropriate use | Function(s) | Switch case – game mode |  |
| Well-structured   * Mostly * All   Indentation  Intuitive order | Well factored – procedures/parameters  names  No duplication  Easily read | Comments where necessary  Largely self-documented code | Enum set – game modes | Concise efficient code |

Further Observations:

Good : : multiple classes and instances interacting. Playable good looking game.

Bad: refactor setup etc – indentation. Use constants for gameModes, or better an enum set.

Overall : Lots of effort – well done!

Grade:82