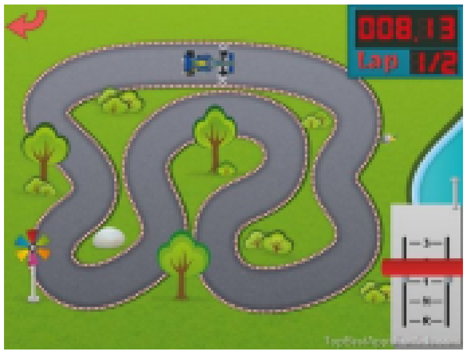
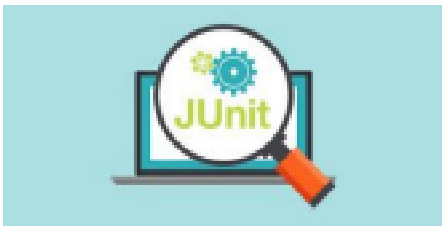
CSE201 Advanced Programming





Your racing game application is working perfectly! Now the gaming company wants you to implement the next version of this game that **must use I/O Streams, Serialization, Deserialization, and JUnit Testing.** Just in case if you did not submit Assignment-5, you can still attempt this Assignment-6 but you will have to first implement Assignment-5 before you start Assignment-6 implementation (using someone else's code is plagiarism!). However, we will only evaluate the code specific to Assignment-6 but not Assignment-5.

In this Version 2.0 of your racing application, you have to extend your implementation to make it more user-friendly and to ensure that the system is foolproof. These new set of requirements from the gaming company are as following:

1. Your race track is now composed of 3 intermediate points: a) 25% of total tiles are covered, b) 50% of total tiles are covered, and c) 75% of total tiles are covered. Whenever either of these 3 checkpoints are reached, your implementation should ask the user if he/she wants to **continue** or **save** the game. If a user selects the **save** option then your implementation should save the current state of the game on disk and then **exit instantly**.
    a. Assumption is that the race track is not too small, such that the player does not move past the second checkpoint directly without even stopping near first checkpoint. There is no hard limit to the track size, but during the demo we will check the save options at least two checkpoints.
2. The player should be able to load a saved game from the disk. The game would simply ask the player Name to load the correct game and resume from the saved state. After resuming, if still some intermediate checkpoints are encountered then Save option should be provided as mentioned in point-1 above. However, this time you can simply delete the earlier saved session and only save the current session for this player login.

3. Version 2.0 platform should be robust to corner cases and should never crash. JUnit based unit testing is a good way of testing your platform effectively and comprehensively. You have to design a **JUnit Test Suite** for your gaming system that contains different JUnit test cases for testing different functionality in Version 2.0 as mentioned below:
    a. Test Cases to validate user defined Exceptions mentioned in Version 1.0. Note that each test cases should be independent of each other, and for this you would have to make some changes to your existing code to accommodate the testing.
        1. SnakeBiteException.
        2. VultureBiteException.
        3. CricketBiteException.
        4. TrampolineException.
        5. GameWinnerException.

    b. Test Case related to Save option in Version 2.0:
        1. Test the serialization of game at first checkpoint.

No test case is provided for this assignment.