Graduation Project Proposal

QAL Gamification: Implementing the Game Mode in QAL

QAL (Quantum Algorithms Lab) is an innovative visual interactive app that is currently under development for researching and teaching quantum algorithms and related mathematical fields. For more info on QAL, check https://q-info.github.io/QAL-Lite.

Abstractly, a game can be viewed as an app or gadget that offers a goal or target configuration that the user/player is challenged to reach from an initial configuration, in a step-by-step fashion, following a set of rules—with a very wide variation in these rules, which define the dynamics of the game—and usually requiring (or rewarding) using the least number of steps.[1]

QAL currently has rudimentary support for a Game Mode. QAL's Game Mode offers the QAL user (the "gamer" or the "player") the opportunity to play one of several games that can be modeled by the math underlying quantum algorithms, such as linear algebra and group theory.

As in many games, the Game Mode of QAL will offer a time-constrained mode ('competition mode') and a non-time-constrained mode useful for training and understanding ('training mode'). During training, once an initial algorithm is reached using QAL, it can usually be improved upon. As a general strategy, the player should use the training mode of QAL's Game Mode to work out a *correct* algorithm (i.e., one that successfully reaches the final position from the initial position), then try to improve on that initial algorithm (e.g., shorten it) to come up with the final (possibly optimal/best) algorithm. Once players are confident about assimilating the game's dynamics, they should test their skills against the competition mode of QAL's Game Mode.

Project Description: In this graduation project students will be responsible of putting their programming skills and web app development skills (using JavaScript, TypeScript, …, etc.) towards providing a simple first-cut implementation of the Game Mode in QAL, by adding QAL support for two (or more) of the games mentioned above using the underlying data structures (e.g., for game state and game evolution) already present in QAL.

Team Size: 2-3 members.

Main Technologies: JavaScript. (Other technologies are optional and, if needed, can be picked up quickly during the development of the graduation project.)

Prerequisites: Excellent programming skills. Good knowledge of general math, particularly of linear algebra, is a plus, but not absolutely necessary.

[1] More precisely, a "single-player mathematical game" challenges the player to reach a 'goal state' or 'goal position' from an 'initial state' or 'initial position,' by requiring the player to devise an 'algorithm' (i.e., a sequence of "allowed/legal moves/steps") to follow. Finding shorter algorithms (i.e., sequences of steps) mean players are better at playing the game. Finding the shortest algorithm/route (to reach the goal position from the initial position) means finding the best (a.k.a., 'optimal') solution to the specific game (or 'problem' or 'puzzle') at hand.

ToyProofs provides a very simple example of a visual mathematical game. Other similar games relevant to QAL include simple ones such as Rubik's Cube, the Pocket Cube (the 2x2x2 Rubik's Cube), Lights Out, MeQanic (an Apple game), Hello Quantum (an Apple and Android game, from IBM), QLogic (an Android game, from paltangames.com), Tap-to-On (an Android game), Quator/Quantum, RedAll (two mobile games from pilesarts.com), but also more advanced ones such as Light Bot, Mekorama, and Scratch … as well as many, many other combination games, permutation puzzles, "coding apps," and toys. Two noteworthy textual — i.e., not strongly visual — "proof assistants" that, nevertheless, have a "progaming" (or game-like) *feel* to them are Proof Designer and Lean. (Copyright © 2009-2024 Moez A. AbdelGawad. The term 'progaming' is a carving from 'proving', 'gaming', 'programming', and, to a lesser extent, 'professional'. The term means 'gaming that has a programming feel to it, and vice versa'.)

Frameworks: QAL is currently implemented as a client-side web app that uses few simple libraries and frameworks (e.g., well-known JavaScript libraries such as jQuery, jQueryUI, … etc.). How these libraries are used in QAL can be explained to the students.

(If students are interested in making major changes to the underlying frameworks used in QAL, e.g., to make QAL use Unity, Node.js, React.js, Vue.js, Anime.js, or other frameworks and libraries, suggestions of such changes are welcome and will be acceptable as long as *strong arguments* for the changes are presented.)

More Details: Contact moez@alexu.edu.eg or moez@cs.rice.edu.