Quackle – Making an open source crossword game in C++ for fun and profit

St. Louis C++ Meetup

January 8, 2020

John Fultz

About Quackle

- Developed originally by Jason Katz-Brown and John O'Laughlin, two expert players in the U.S. Scrabble® tournament circuit at the time
- Provides
  - User interface
  - Static play analysis
  - Simulator for candidate plays
  - A variety of computer players powered by combinations of static and simulation analysis
  - Tools for configuring boards, dictionaries, letter distributions
- Released in 2006, and continuously maintained since then
- Since its release, has been the dominant tool used by Scrabble® players at all levels
- Beat a world champion player 3-2 in an official tournament setting in 2006
- Successor to Maven, the most significant crossword game engine of the 90s

My history with the project

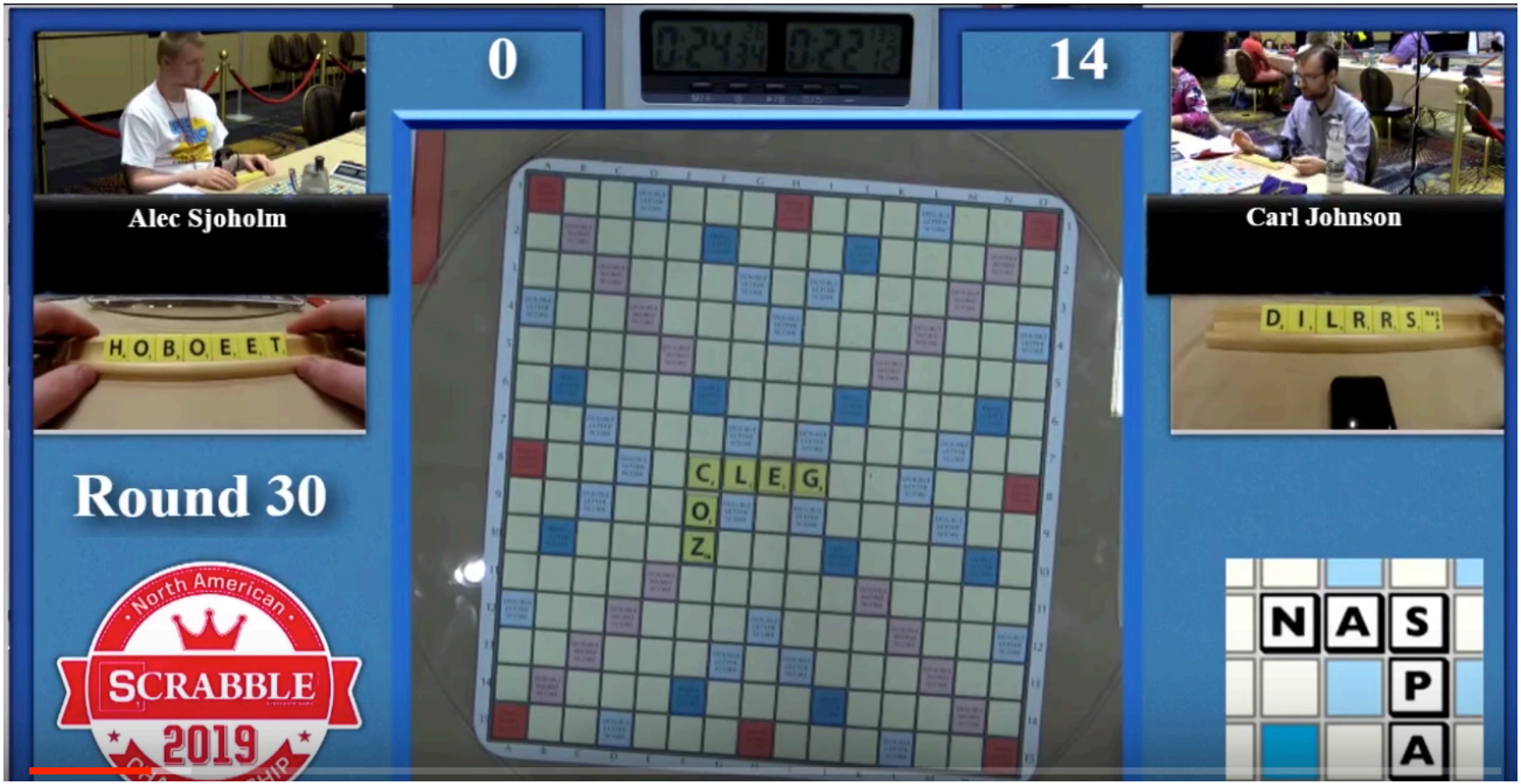
- Joined very shortly after the initial release
- Created the Windows port
- Created the board configurator
- Worked with Jason & John to release more versions in the 2000s
- In the 2010s, as Jason & John stopped contributing, continued to update and release versions, taking over maintenance of macOS and Linux versions
- Begun adding major new features in response to external events and just stuff I wanted to do

The architecture of Quackle

- libQuackle – the Quackle game engine, includes:
  - Data structures for boards, players, moves, dictionaries
  - Evaluation and simulation engine
  - Computer player implementations
  - SWIG bindings (and working on command-line tools)
- libQuackleIO – I/O library, requires libQuackle
  - Writes/reads GCG (game) files
  - Creates various dictionary files
  - Simulation reporters, writing to txt and html
  - Requires (unfortunately) Qt
- Quacker – the Quackle GUI
  - Based on Qt5
  - Allows selection of dictionaries, alphabets, boards

- Originally written in C++03, gradually migrating to C++14

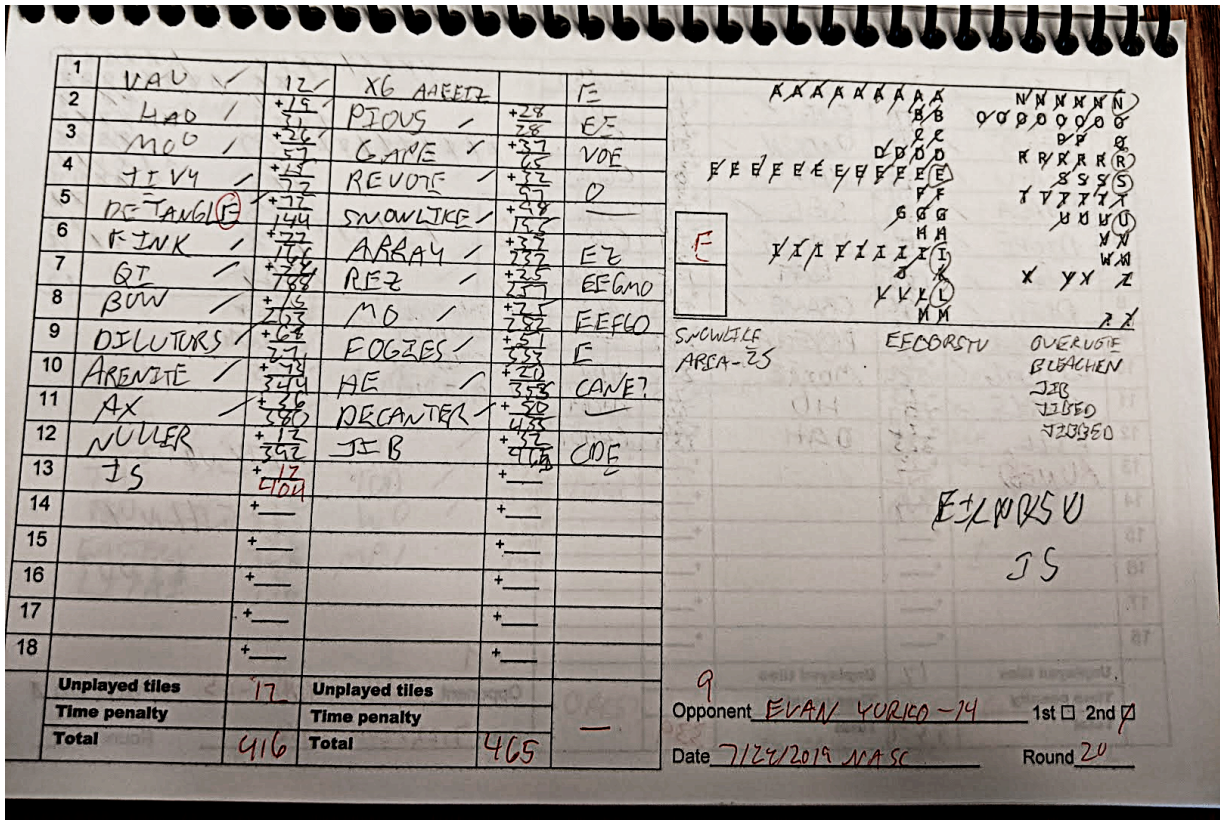
How tournament games are played – Two players



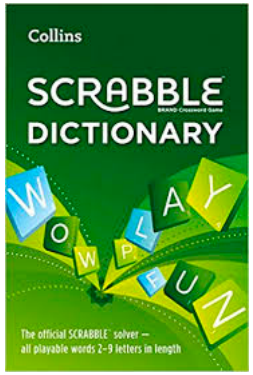
How tournament games are played – With a clock

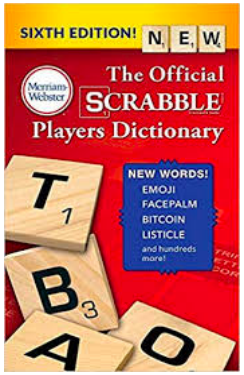
How tournament games are played – Players maintain scoresheets

- Good players may take notes
- Good players always track tiles
- Enough information to recreate the game for computer analysis



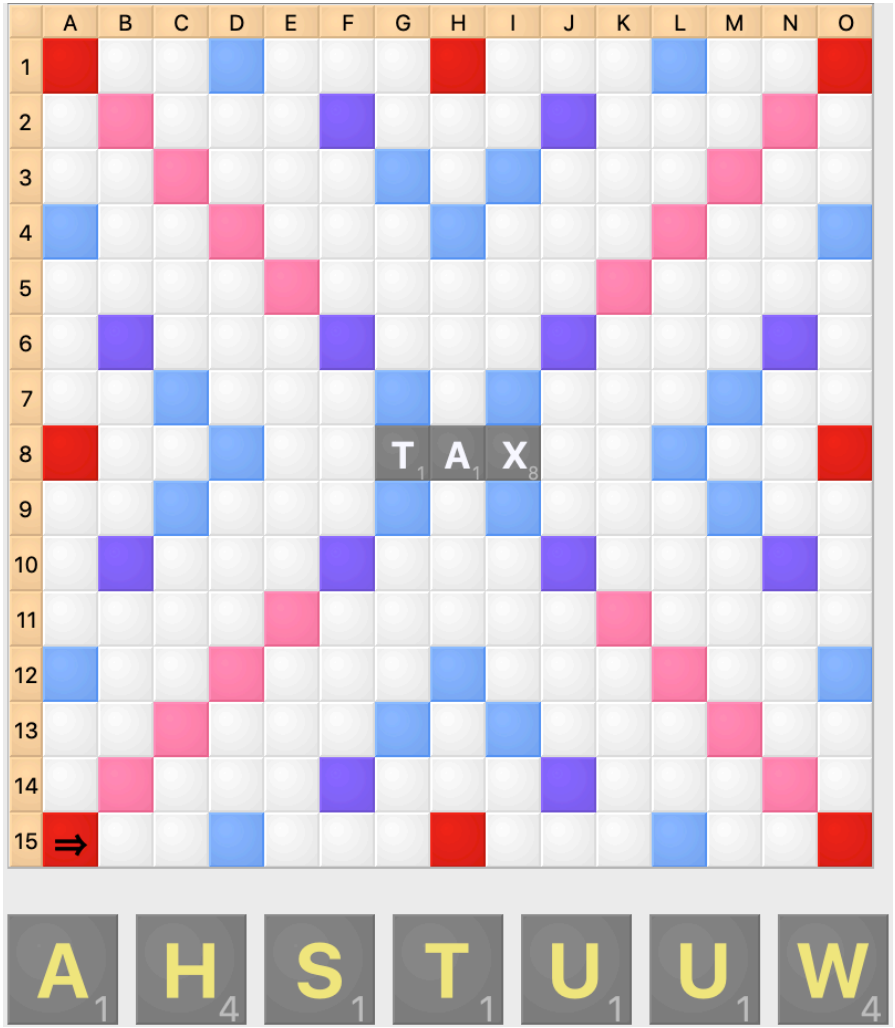
How tournament games are played – Authoritative dictionary





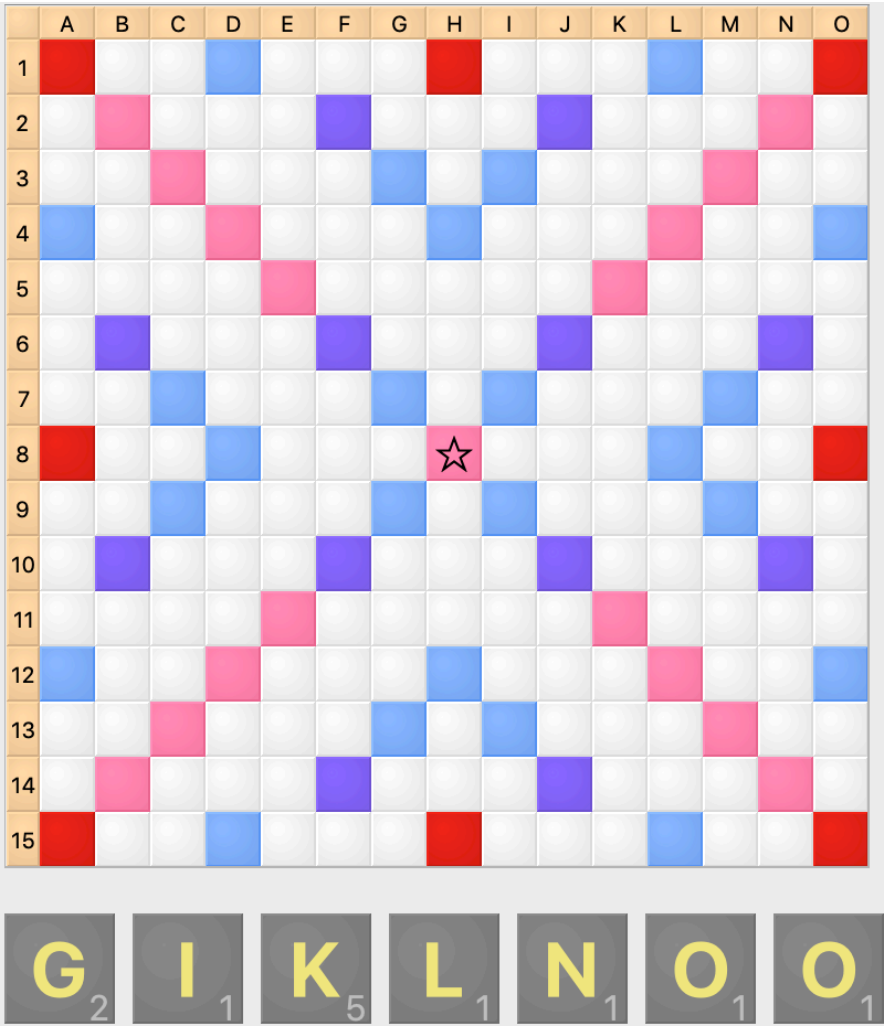
- Dictionaries dictated by a central body
- Dominant English dictionaries
  - Collins dictionary for international play
  - NASPA Word List for North American play
  - "School list", which usually coincides with the Official Scrabble Player's Dictionary – like NASPA Word List, but without the "naughty" words
- Dictionaries are updated routinely
- Standard computer word lists are…sometimes…provided for software consumption

How to play crossword games well
Know the words

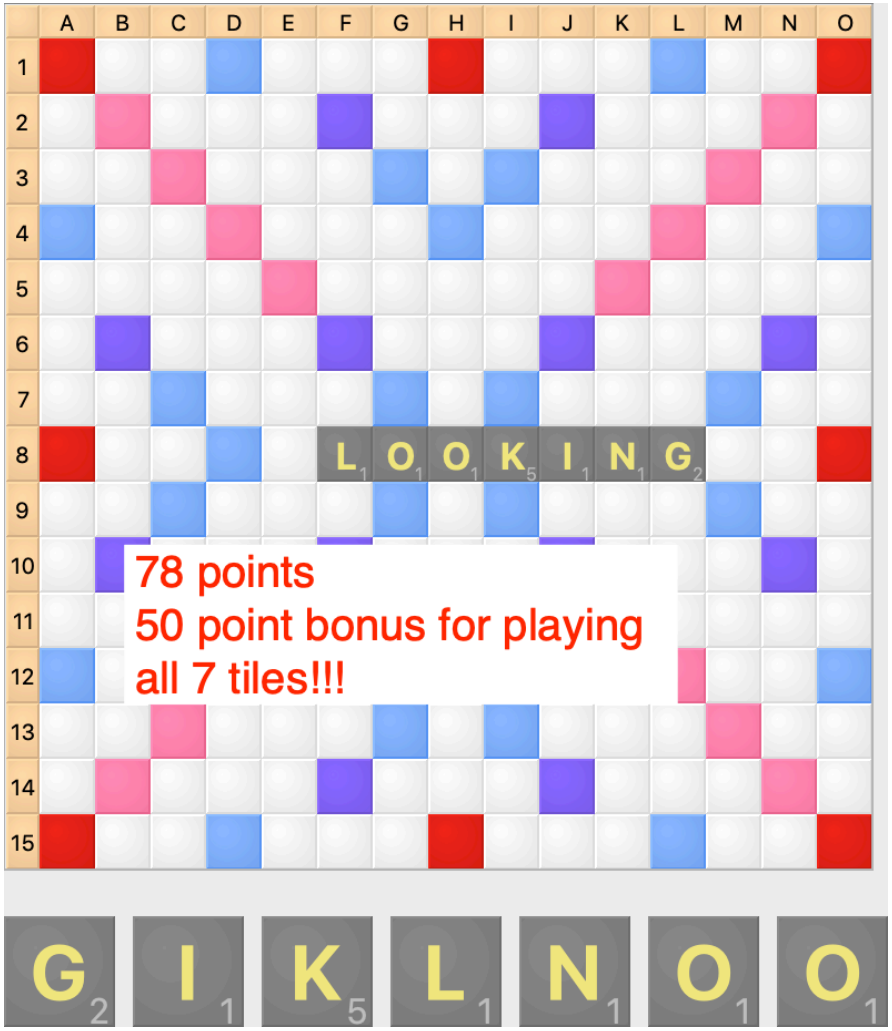


How to play crossword games well
Know the words

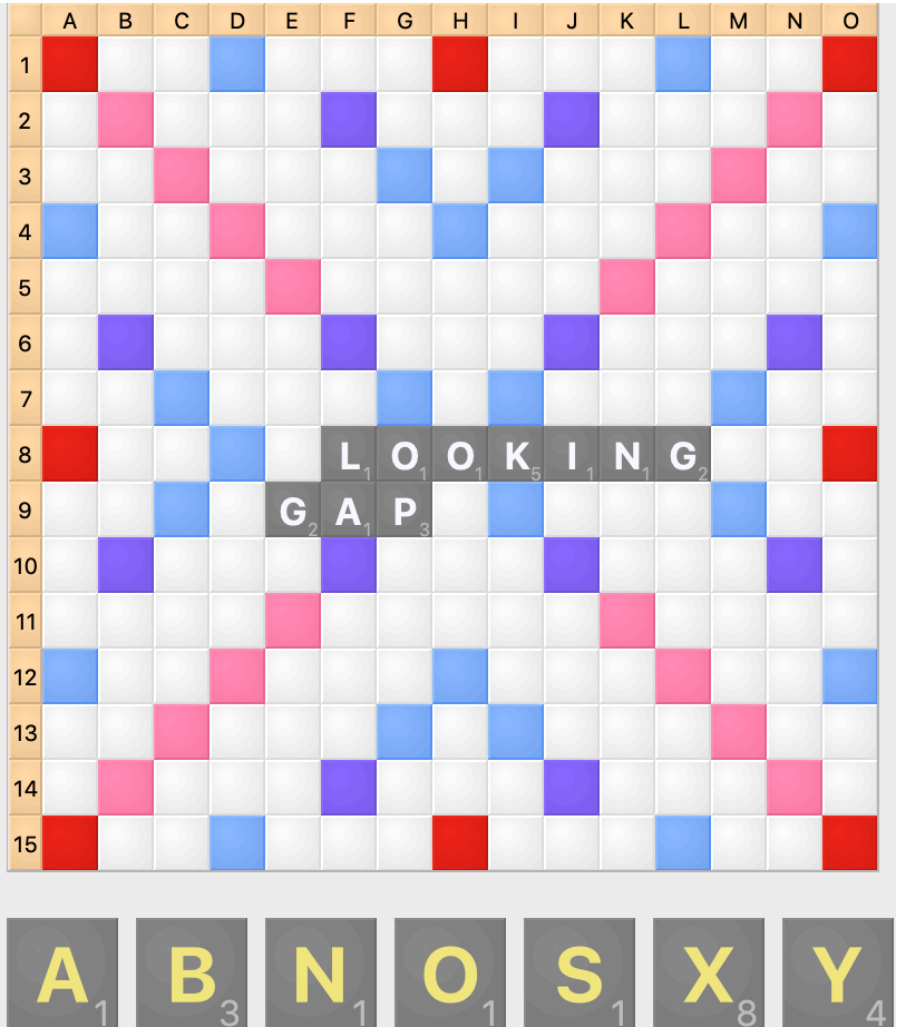
How to play crossword games well
Playing bingos



How to play crossword games well
Playing bingos



How to play crossword games well
Using bonus squares well



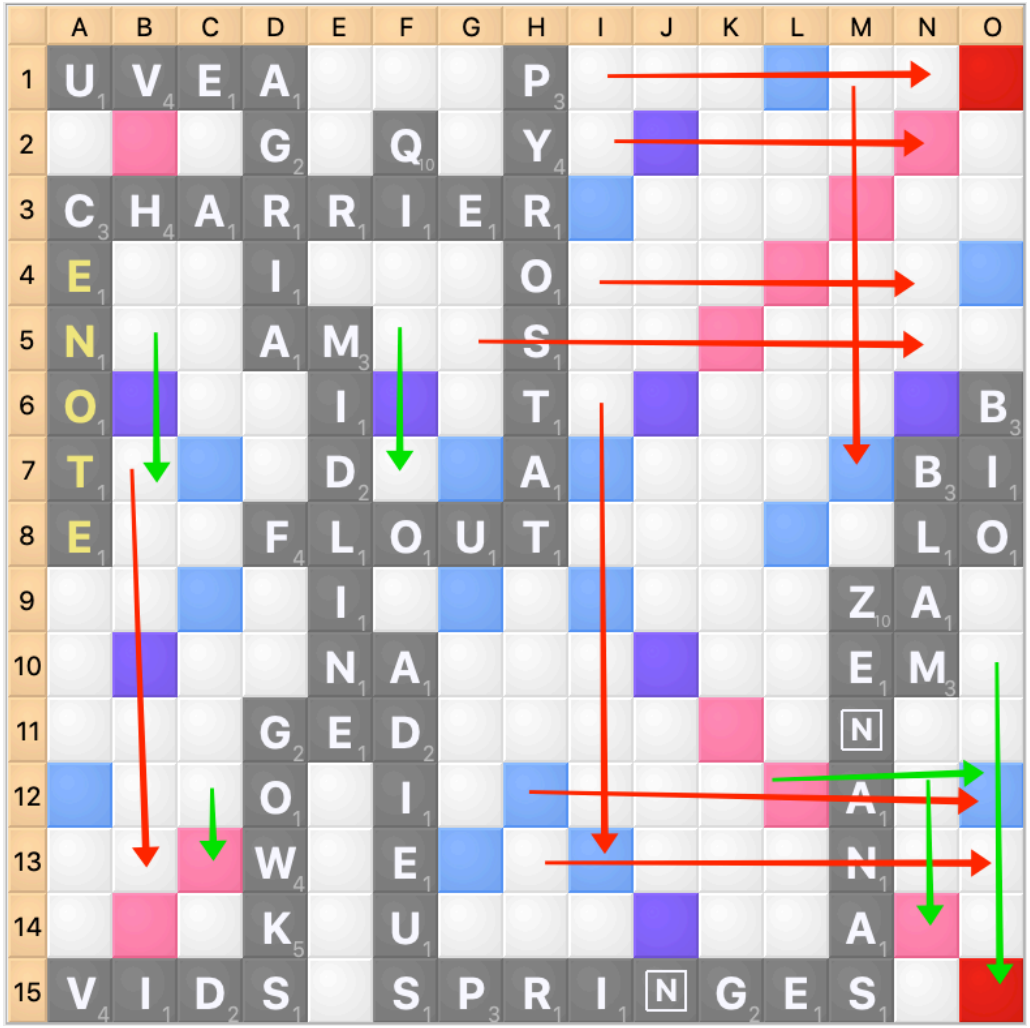
How to play crossword games well
Using bonus squares well

How to play crossword games well
“Board vision”

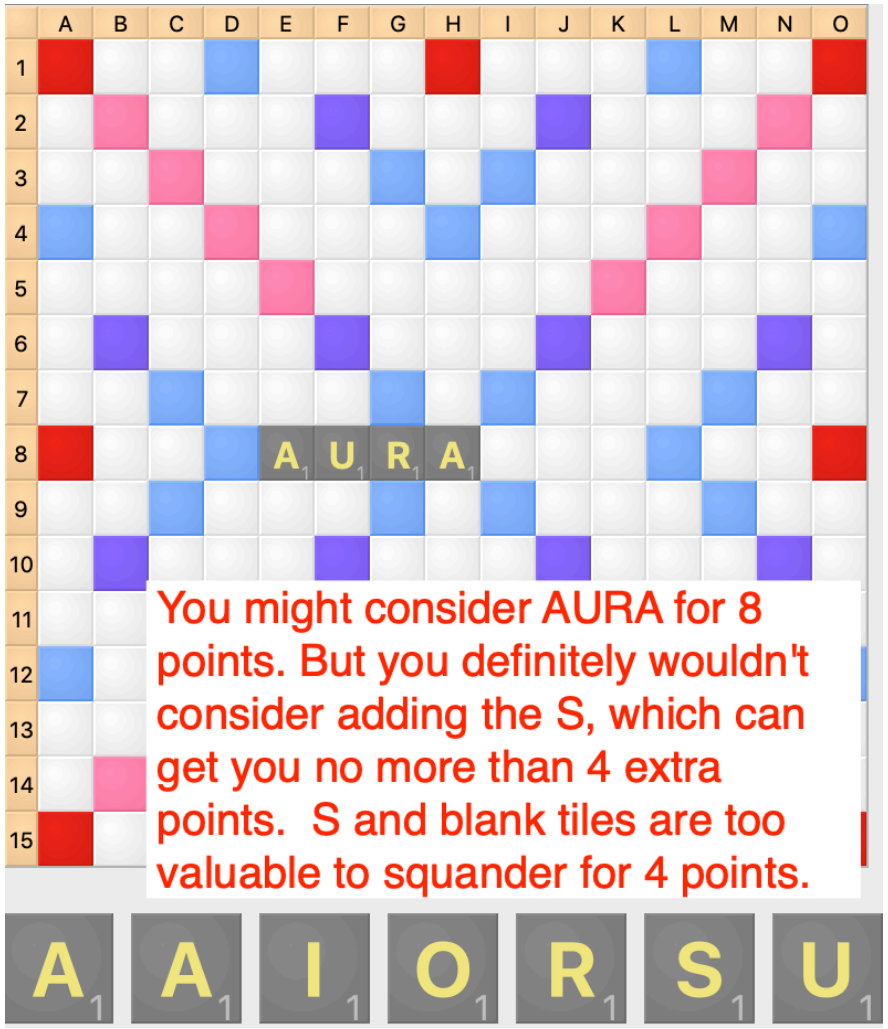
- Being able to analyze a board and finding…
  - “Bingo lanes” – places where a 7-letter play might happen
  - Bonus tiles that having significant scoring opportunity
  - Combinations of bonus tiles that can be used in one play
  - Any other “interesting” scoring opportunities

The following slide notes bingo lanes in red, scoring opportunities in green. A seasoned player will quickly be able to assess these for their own use or determine whether they represent a threat if left open for their opponent to use.

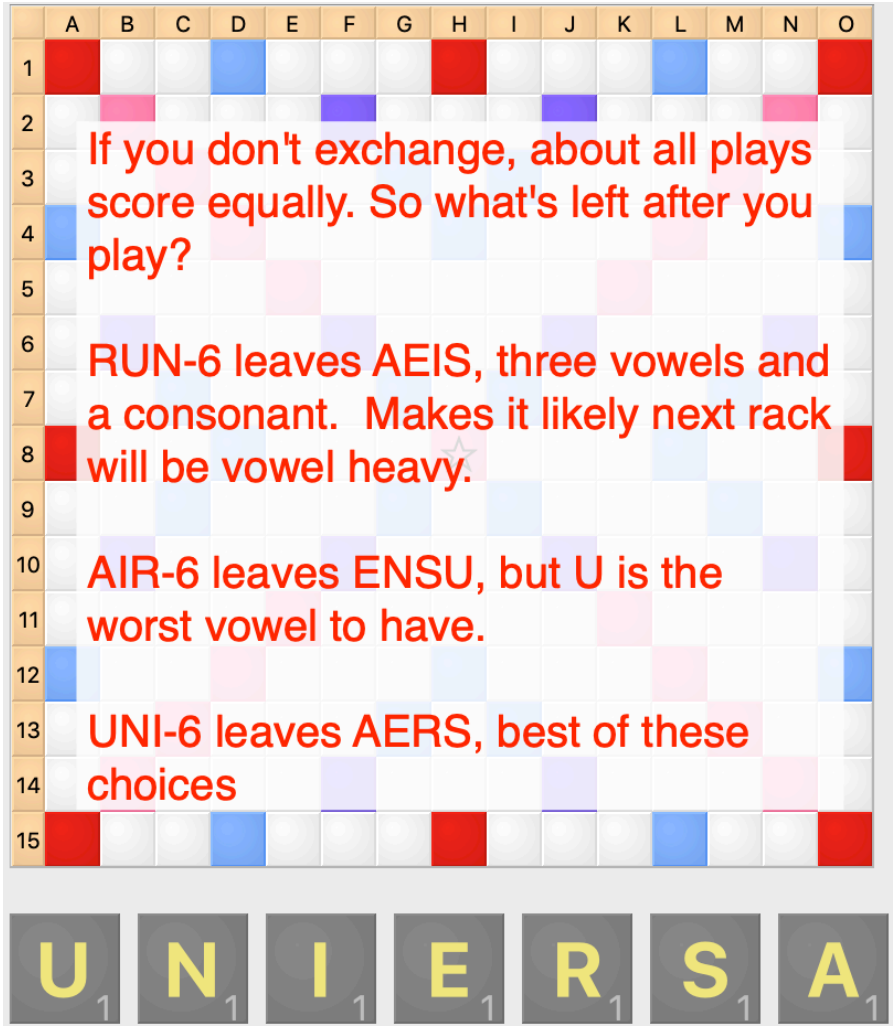
How to play crossword games well
“Board vision”



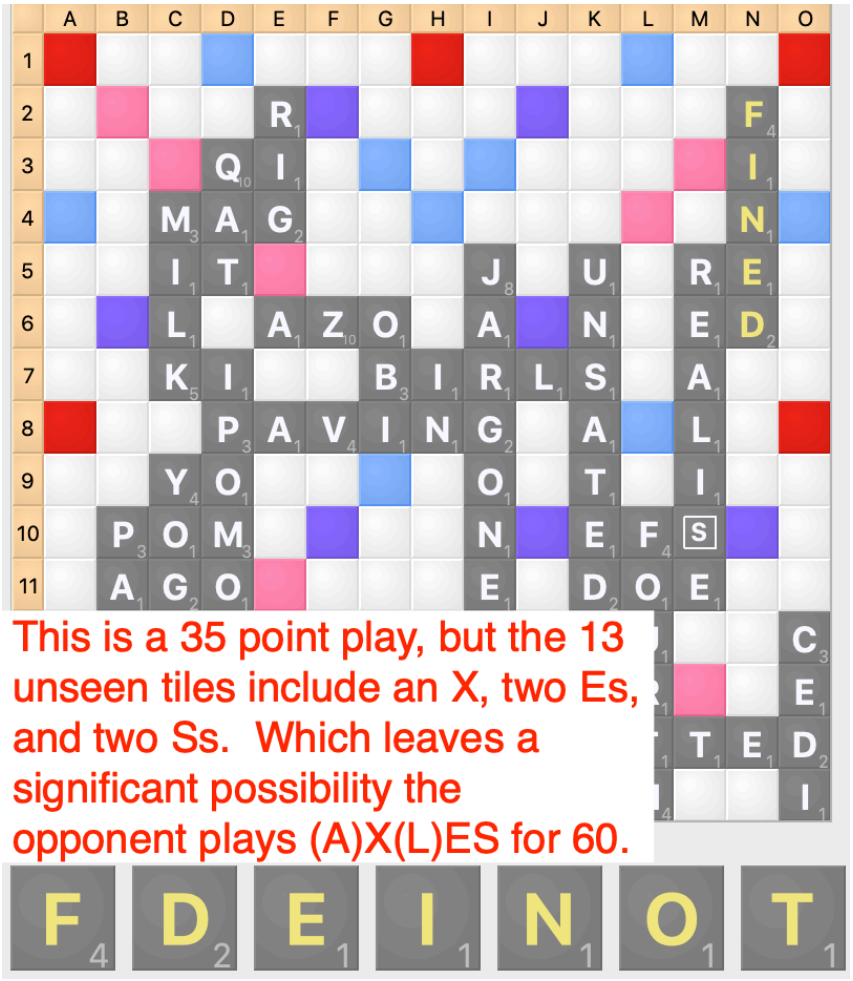
How to play crossword games well
Don’t squander great tiles



How to play crossword games well
Consider your “leave”



How to play crossword games well
Consider your opponent's rack



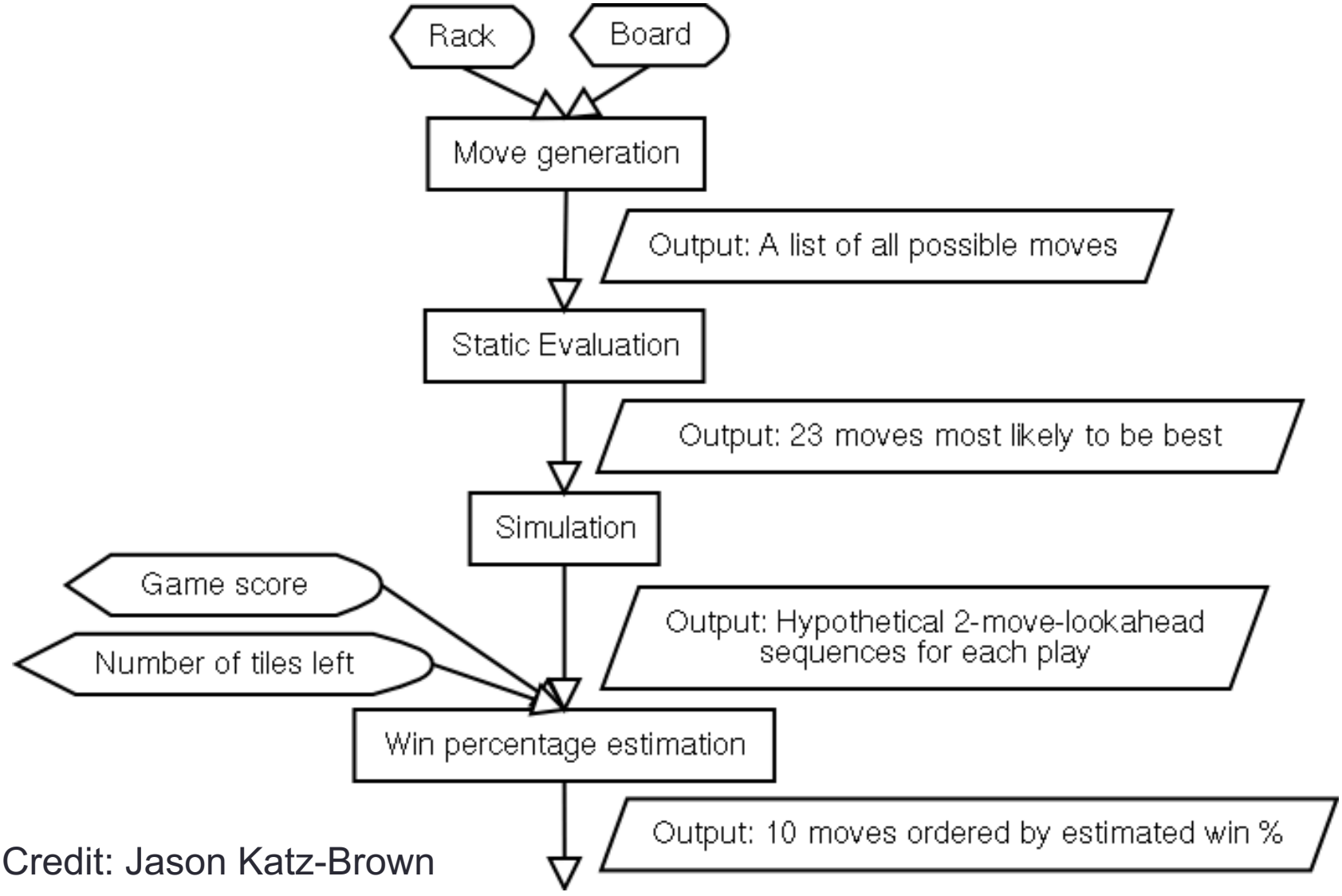
Algorithms – Static move evaluation

- List all the moves
- Assign “equity” scores to the moves based on
  - Actual score of the play
  - Score assigned to the “value” of the leave
    - Individual tiles may have values
    - Groups of tiles may have a “synergy” value
    - Consonant/vowel ratio affects the score
    - Or, to improve speed, a “superleaves” table may be precomputed with all possible leave values for a given dictionary

Algorithms – Simulation

- Start with a discrete list of moves
- Ply 1 – Opponent draws 7 random tiles, and makes best statically evaluated play
- Ply 2 – Player replenishes random tiles and makes best statically evaluated play
- Repeat for as many plies as requested
- Resulting scores plus final leaves are recorded; if the simulation ended in a win/loss, that is recorded
- Win %age is computed based upon actual wins/losses for those sims that ended the game, and a lookup table of point spread vs. remaining tile count for those that didn't
- Simulation is nearly embarrassingly parallel, but we only added threading support last year

Algorithms – Computer players



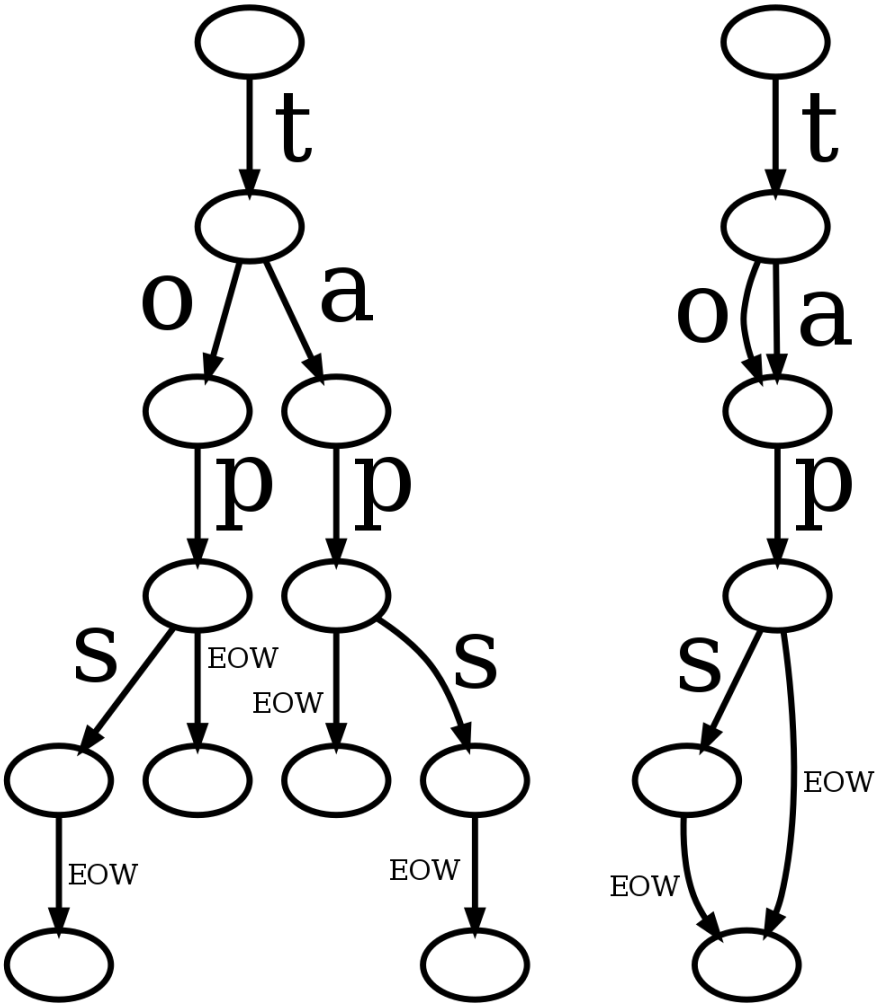
Credit: Jason Katz-Brown

Algorithms – Computer players

- Computer players can't effectively sim all the plays in most scenarios
- Different computer players have different algorithms for how many candidates to simulate, and how many simulations to run
- However, in the endgame play, a smart opponent can sneak through a low-ranking statically evaluated play, so endgame must be considered differently

- More info here: http://people.csail.mit.edu/jasonkb/quackle/doc/how_quackle_plays_scrabble.html

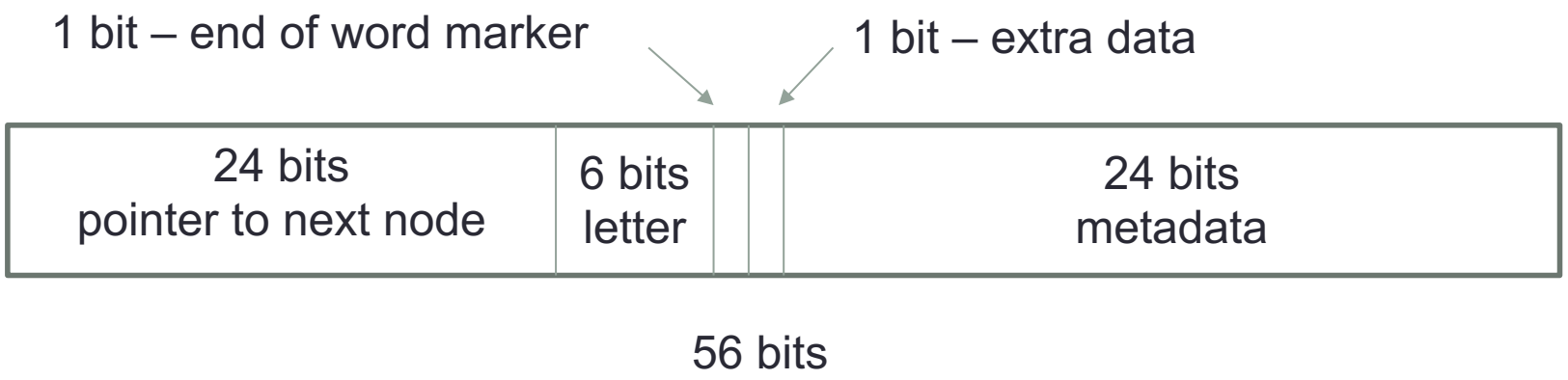
Data structures – DAWG



Credit: Wikipedia

Data structures – DAWG

- The DAWG (Direct Acyclic Word Graph) is used by many programs that want to efficiently store a word list with a fast lookup
- Also known as a “trie” (pronunciation not universally agreed upon)
- Quackle’s DAWG supports 2^24 nodes, 2^6-1 letters



Interestingly, older versions of Quackle supported only 5 bits per letter, but this isn’t enough in some languages.

Data structures – GADDAG

- A novel structure invented by Steven Gordon in 1994
- Like a DAWG, but allows you to start with any letter in the word and work your way outward
- Highly redundant data structure, but finding plays is highly efficient
- GADDAG is a play on DAG
- To quote Wikipedia, the GADDAG will contain direct paths to these strings for the word EXPLAIN:

```
e+xplain
xe+plain
pxe+lain
lpxe+ain
alpxe+in
ialpxe+n
nialpxe
```

Open source – Community

- My guess…Quackle has a dedicated user base in the low thousands
- A few have GitHub accounts and report bugs directly; others send mail to a Yahoo! mail list
- Nine contributors including myself and the two creators
  - Some contribute just data file updates, such as dictionaries
  - Some newbies, and a few very seasoned folks
- Many thankful people in the community
- Some critical, but still thankful
- And one or two haters because Internet

Open source – Intellectual property

- Licensing Quackle as GPL
  - But also sold a non-exclusive license early on to an online game company who wanted to modify and use our game AI
- The SCRABBLE® trademark
  - Many players of the trademarked game use Quackle
  - But we cannot and do not use the trademark to promote Quackle
- Copyright and the game
- Copyright and the word lists
  - Supporting user-added dictionaries
  - We broadly advertise copyrights of bundled dictionaries, and even of user-installed dictionaries if we can identify them
  - Original idea for creating a dictionary hash to identify copyrighted dictionaries had other multiple benefits:
    - Allows users to identify canonical word lists
    - Trivially ensures DAWG and GADDAG are synchronized

Open source – Researchers

- Researchers use Quackle to do various kinds of research on games, or on human and computer intelligence
- scholar.google.com shows at least 15 references to Quackle in scholarly papers
- Some merely reference or cite Quackle, but a few have modified Quackle to support their research
- An example paper on modeling “human-like suboptimal” computer players:

http://reason.cs.uiuc.edu/mdrichar/my_papers/human_scrabble.pdf

Open source – Maintenance challenges

- Moved from Qt3 to Qt4 to Qt5
- Original code base was C++03, now on C++14
- Build system started with scons, moved to qmake, now migrating to cmake
- Evolving internationalization challenges
- Digital signing is not free, and has been modestly painful
- Supporting macOS Catalina required App Notarization, which was hours of work
- Needed to add support for HiDpi
- Beginning to get “dark mode” bugs
- Travis-CI sometimes requires dark magic

Open source – Tools used

- Travis-CI – macOS and Linux builds
- AppVeyor – Windows builds
- GitHub Issues – bug and feature tracking
- GitHub Projects board – organizing issues into a roadmap
- cmake
- Qt – GUI toolkit, encoding support, SHA implementation
- Inno Setup – Windows installer

Open source – Tools I'd *like* to use [when I have the time…]

- Coverity Scan – free static analysis for open source
- clang-format, clang-tidy
- GoogleTest or Catch2 – unit testing