Board Game Game Engine in Haskell

David Cao (davidcao) and Dylan Mann (mannd)

**Project Goal**:The goal of this project is to write a board game engine. The engine will provide developers with a simple to use framework to write any sort of board game, such as something as simple as Tic Tac Toe to some as complex as Monopoly or The Game of Life. The engine will allow developers to both customize the graphics and integrate their game logic seamlessly. We plan to demonstrate the game engine by building a non-trivial game with it, such as Settlers of Catan.

**Typical Use Case**: We will use the example of Settlers of Catan throughout the use case. When programming a board game, there are several different parts that need to be considered. First is the state representation of the game, i.e. how will the programmer maintain the state of the game internally. Second is the graphical interface, i.e. how will the game be rendered, what graphics engine should we use. Third is handling player turns, gameplay, and win conditions. And lastly is the possibility of computer controlled players, or AI.

Some of the design was taken from <https://github.com/TGOlson/deterministic-game-engine>

**Moving Parts**:

* Loosely inspired by MVC model
* Idea is program runs an event loop that waits for specific actions, and updates the game state and redraws the game board accordingly
* Note that the below methods are subject to change!
* GameEngine is a class that requires several methods to be implemented
  + setup :: List Player -> List Action -> GameState
    - Returns the initial state of the board
  + nextPlayer :: GameState -> Player
  + winner :: GameState -> Maybe Player
    - Called automatically after each action, should be defined to check for win conditions
* Graphics engine integration
  + drawOnAction :: GameState -> IO()
    - Draws board for a given GameState, automatically called after an action is taken
  + drawOnIO :: GameState -> IO -> IO()
    - Draws board for a given IO input, i.e. mouse moved or clicked
    - Note that this method is intended for graphical eye candy, i.e. highlight on hover.
* Provided methods
  + handleAction :: GameState -> [Actions] -> GameState
    - Handles the first action to happen in the list
    - Use case will be something like `newState <- handleAction oldState`
* State representation (up to developer, but we provide nice things to update the state with)
  + Methods to update state based on event that happened, i.e. this clicked or this entered
    - Multiple types of prompts/methods, click for a button, prompt w multiple entries, or straight text entry

**Effort Budget**:

1. Implement basic types and classes: 2 hours
2. Pick a graphics engine to use and plan how to link it: 4 hours
3. Implement backend things, i.e. handleAction and linking together the graphics engine with our own game engine: 6 hours
4. Step back and look at our framework from a developer’s POV, redesign things if necessary: 5 hours
5. Make Catan using our framework (also use this as a method to prototype our framework, fixing things along the way)! 6-10 hours