Software project, 2014 - Semester A : implementation description

Submitted by: David Lifshitz (davidl2), Guy Daich (guydaich)

Implementation Overview

The main function loads a start menu, using appropriate window_drawing and button_actions infrastructure. in each menu, user clicks are interpreted and acted upon by relevant handlers, that change a global ui_tree, and re-draw it.

Eventually, a game is either chosen or loaded, causing a global general_game object, to be created - instantiating an abstract class with some specific game. This abstract class's panel_function, is used to create a game specific UI-subtree, that is inserted in the main UI-tree, and drawn.

All game logic and operations (interpreting user moves, checking for victory, making moves and updating the ui-tree, saving the game to file, etc.) are made against the abstract class. The minimax is called with relevant game parameters, and the game-specific state_children function. alpha-beta pruning is used for minimal tree construction and on-the-fly deletion, creating each level with the supplied children function.

Controls are drawn as specified in the description. panels are used for logical and physical partitioning, with controls blitted onto (and offsetted, cropped) according to their owning_surface - their direct panel ancestor. panels, labels and buttons may be drawn with a background image or RGB surface, the second being a default, and transparency is optional. SDL_TTF is used to create and blit textual surfaces. buttons use a simple, middle-aligned captions, and labels may have captions drawn with line-breaks (effectively, a bounded array of surfaces).

In every window, the SDL loop waits for on_click events. These are interpreted according to final control positions (offsetted by panels). The relevant control's (button's) handler, is called. Menu buttons, usually cause some screen change. The game board and pieces are also buttons, where a click is handled by a game specific handler - determining the move's validity, and redrawing the game sub-tree. When the game is paused, both player and AI cannot make moves, until the game is unpaused. When the AI plays first, the game begins paused, and an AI v. AI game requires unpausing for each move. The current player is displayed, and victory is either highligted or announced.

There are three types of windows - menus (generic selection from a set of levels - e.g. difficulties, or preset like initial menu), the game window - consists of a both a button panel and a game panel, and notification windows - asking the user for Yes\No or notifying him of some issue (only OK), with window transitions implemented as UI-tree freeing, rebuilding, and redrawing.

All errors: SDL errors, allocation errors, missing files, failures in tree (UI or minimax) construction, are treated as fatal errors. Appropriate Errors are sent to the console, in each level of the game's implementation, and resources (trees and their node contents - game states or controls), are freed, SDL and TTF are closed, and the program terminates. Corrupt or missing save files, are reported to the user

via a message window, and the game continues. We regard a file as corrupt if there is no game name, if there are unexpected piece characters, or if there are more pieces than possible.

Descriptions of modules and classes

Bellow is a more detailed review of functionalities of classes.

UI infrastructure:

- **controls.h**
  - defines a general control factory: a struct holding all necessary data - position, size, associated images, captions, surfaces and so on. it also defines certain behaviors, like an "on-click" handler, and a control-specific drawing function.
  - data structures for the UI tree storing (a linked list implementation of a tree including elements and lists, a lighter version of the minimax tree structures, for convenience).
- **controls.c**
  - every control (window, panel, label, button) differs in the way it is initialized. we hold different initializes (new_label, etc).
  - every control is drawn differently, and has a specific drawing function.
  - the maintenance of control data structures - creating, linking and freeing.
  - several misc functions for control handling.

Game Logic and GUI:

- **connect4_bl.c, connect4_bl.h**
  - implements game functions required in project description: init, state children, score, etc.
  - implements additional functions such as victory check, game over check, move handle (validation, game object update, UI tree update), minimax call handle. There are used in our general_game object.
- **connect4_ui.c , connect4_ui.h**
  - implements the panel function required in project description.
  - implements ui victory handle - find winner, winning move, and update GUI to highlight it.
- **ttc_bl.c, ttc_bl.h, ttc_ui.c, ttc_ui.h** - much the same as the above, classes for handling Tic Tac Toe logic and UI.
- **reversi_bl.c, reversi _bl.h, reversi _ui.c, reversi _ui.h** - much the same as the above, classes for handling Reversi logic and UI.

General UI

- **buttonActions.c, buttonActions.h** - on_click handlers for all buttons in the game. Has several types of handlers:
  - window change handlers - open new windows instead of current one, and managing game state, ui tree re-draws, etc.

    - game logic handlers - for Human involved games (not AI vs. AI), when a player makes a move, we call game-specific handlers to modify game state, and ui tree. we also call minimax, for computer moves.
- **windowsDrawing.c, windowsDrawing.h** - responsible for drawing of all UI windows
    - the game window, is initialized with a get_default_ui_tree(), and then has a game-specific sub-tree added, according to chosen game.
    - run_window is used for drawing new windows all the menus for selecting game options, game types, etc. init_choice_window constructs all parameters for generic multiple selection windows (buttons for several level, etc). start_window handles ui_tree creation for main menu. choice_window - handles drawing of all choice windows, once their tree was built with with a relevan function. notification_window construct ui tree for notification windows. question window - draws and runs the constructed notification windows.
- **windowsDrawing.h**
    - notification_type enum - used to represent different types of notification windows - a yes/no window, or a simple notification.
    - user_selection enum - represents the sort of menu to move to - game selection, load, etc.
- **game_enum.h** - simply represents all games across several classes, when a game type is passed as a parameter.
- **genereal_game.c, general_game.h** - an abstract class, that consists of some game members (board, dimensions, difficulty, etc.) and game methods (implemented as function pointers).
- **game.c, game.h** - the program's main function, responsible for init of libraries, and running the main SDL loop.

<u>minimax</u>

- **minimax.h**
    - minimax data structures - original nodes, elements and list structures supplied for minimax in exercise 3. they are used by both by minimax, and game logic classes for creation of state children.
    - piece enum - defines the player (1, 2 or none) ,associated with a piece on the board
- **minimax.c**
    - data-structure implementation - allocation, linking, freeing, etc.
    - minimax tree construction - the alpha-beta function, receives several parameters, one of which is a pointer to a game-specific "state-children" function. ask david

<u>save games</u>

- **save_game.c, save_game.h**
    - handles the saving of a given game state to a file
    - handles reading of a saved game from files, to appropriate game boards.