Dev Documentation for Conway's Game of Life Project

Files

- **main.c**: Manages the main game loop, user input, and window rendering.
- **functions.c**: Contains every function for the game.
- **function.h**: Header file for declaration of every function and including every library.

Data Structures

- `grid` and `old_grid` : 2D arrays (`unsigned short**`) representing the game board. Each element stores the state of a cell: `0` for untouched, `1` for alive, `2` for dead.
- `sfRenderWindow*` : Pointer for the main game window where the game grid is displayed.
- `sfText*` : Pointer for text which will be rendered on the window (instructions, population).
- `sfRectangleShape*` : Pointer for the only square (cell) on the grid. This rectangle will be displayed on the main game window in multiple location, so the user could see the game board.
- `sfClock*` : Timer for tracking elapsed time for frame updates.

Algorithms

1. **Game State Transition**:
   - Every cell in the grid has its state determined by its neighbours in the previous state of the board. The neighbour count is checked by iterating over the `old_grid` in function `count_neighbours` . The result is stored in the `grid` with function `kill_or_revive_square` , which is then displayed by function `dysplay_main_window` and copied into `old_grid` by function `copy_grid` . The whole process is inside function `next_frame` .
   - The game state is updated either on user command by manual step or in real-time while the game is running.
2. **User Interaction**:

- The game can be paused, resumed, and changed by the user with keyboard or mouse inputs.
- Keyboard inputs include:
- `TAB` - show/hide instructions;
- `Space` - start/pause the game;
- `LShift` - download `.txt` board from your device;
- `Enter` - upload `.txt` current board to your device;
- `Backspace` - clear the board;
- `Escape` - close the game.

3. **Memory Management**:
   - Dynamic memory is used for the game grid (`grid`) and the previous state grid (`old_grid`), because otherwise it would take a lot of stack memory due to the possible sizes of the game grid.
   - Also dynamic memory is used for almost every structure and union from CSFML library in order to not fill in the stack.
   - Memory is allocated at the beginning of the program and freed upon exit either with `free` function or `destroy` function from CSFML library.
4. **Grid Rendering**:
   - The grid is drawn as a collection of squares using CSFML's `sfRectangleShape` . Each square's colour indicates whether a cell is untouched, alive or dead.
   - The window is updated every 50 milliseconds to achieve a framerate of 20 fps.

Function List

1. `main` :
   - **Description**: Sets up the game window, handles user input for the size of the board, allocates and frees memory, and runs the main game loop.
   - **Parameters**: None.
   - **Returns**: `int` (`0` if successful).
2. `display_main_window` :
   - **Description**: Displays the current state of the grid on the window, including the population count and instructions if necessary.
   - **Parameters**:
     - `sfRenderWindow* main_window` : The main game window.
     - `sfRectangleShape* square` : A pointer to the rectangle shape for drawing the cells.

- `unsigned short** grid` : The current game grid.
- `const unsigned short AMOUNT` : Number of squares in each row/column.
- `const unsigned short SIDE_OF_SQUARE` : Side length of each square.
- `sfText* population` : Text for displaying population count.
- `sfText* instructions` : Text for displaying instructions.
- `unsigned short status_of_instructions` : Determines if instructions should be shown.

- **Returns**: `void`

3. `next_frame` :
   - **Description**: Advances the game by one step by applying Conway's Game of Life rules to the grid.
   - **Parameters**:
     - Same as `display_main_window` with additional parameter `unsigned short** old_grid` in order to count the neighbours of each cell on the current state of board while changing the `grid` .
   - **Returns**: `void`
   - **Notes:**
     Function `next_frame` is used every time when we need to show `main_window` with the next state of the board. This function uses every other function in its body.
4. `load_grid` :

   - **Description**: Loads a saved game state from a `.txt` file into the grid.
   - **Parameters**:
     - `unsigned short** grid` : The current game grid.
     - `const unsigned short AMOUNT` : Number of squares in each row/column.
   - **Returns**: `unsigned short` (0 on success, 1 on read error, 2 on file error)
   - **Notes:**
     Structure of the file:
     `number%number%...%number%number$\n` - There are AMOUNT numbers
     `number%number%...%number%number$\n`
     `number%number%...%number%number$\n`
     .
     .
     .
     `number%number%...%number%number$\n` - There are AMOUNT rows

   If the file is not like this, the function will return error code 1.
5. `upload_grid` :
   - **Description**: Saves the current game state into a `.txt` file.

- **Parameters**:
  - Same as `load_grid`.
- **Returns**: `void`
- **Notes:**
  Structure of the file:
  `number%number%...%number%number$\n` - There are AMOUNT numbers
  `number%number%...%number%number$\n`
  `number%number%...%number%number$\n`
  .
  .
  .
  `number%number%...%number%number$\n` - There are AMOUNT rows

The function upload_grid will create a .txt file exactly by this sample.

6. `clear_grid`:
   - **Description**: Resets the grid by setting all cells to the default state.
   - **Parameters**:
     - Same as `load_grid`.
   - **Returns**: `void`
7. `copy_grid`:
   - **Description**: Copies the contents of one grid to another.
   - **Parameters**:
     - `unsigned short** grid`: The grid to copy from.
     - `unsigned short** old_grid`: The grid to copy to.
     - `const unsigned short AMOUNT`: Number of squares in each row/column.
   - **Returns**: `void`
8. `count_neighbours`:
   - **Description:** Count neighbours of each square.
   - **Parameters:**
     - `unsigned short** old_grid`: The grid on which neighbours will be counted.
     - `int row`: The row of the square.
     - `int column`: The column of the square.
     - `const unsigned short AMOUNT`: Number of squares in each row/column.
   - **Returns:** `unsigned short`
   - **Notes:**
     Return of this function will be a parameter for the `kill_or_revive_square` function.
9. `kill_or_revive_square`:

- **Description**: Killing or reviving a square based on the number of neighbours from the previous function.
- **Parameters:**
  - `unsigned short** grid`: The grid on which neighbours will be counted.
  - `int row`: The row of the square.
  - `int column`: The column of the square.
  - `const unsigned short counter`: Number of neighbours of the square.
- **Return:** `void`
- **Notes:**
  Rules of the game for function:
  1. Any live cell with fewer than two live neighbours dies, as if by underpopulation.
  2. Any live cell with two or three live neighbours lives on to the next generation.
  3. Any live cell with more than three live neighbours dies, as if by overpopulation.
  4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

10. `to_string`:

- **Description:** Converts `unsigned short` to `char*` (only for positive integers). This function is used to convert numbers to `char*` in order to show current population and to upload the current state of the board into `.txt`.
- **Parameters:**
  - `unsigned short number`: The number to convert in a `char*`
- **Return:** `char*`
- **Notes:**
  This function allocates memory for char*, so it should be cleaned manually afterwards.

11. `to_int`:

- **Description:** Converts `char*` to `unsigned int` (only for positive integers). This function is used to convert `char*` into numbers in order to load the state of the board from `.txt`.
- **Parameters:**
  - `char*`: The `char*` to convert into a number
- **Return:** `unsigned int`
- **Notes:**
  This function does not free memory from char*, so it should be cleaned manually afterwards.