Hi Alice,
We've heard so much about your work, it's great to have you join the project. The code is hopefully self-explanatory, but we'll run through it for you here.

- EtC team

Installing EtC

This release of Escape The Clocktower requires **cmake** for installation. It also comes bundled with the Allegro library, which must be installed prior to installing the game itself. Cmake can be downloaded freely from https://cmake.org/download/.

The root directory of “Escape-the-clocktower-alpha” contains an “allegro” folder. This folder contains a cmakelists.txt file which can be used to install Allegro. To install using cmake, navigate to ../Escape-the-clocktower-alpha/allegro/ then run **cmake**.

This will configure cmake and create a makefile.mk in the same directory. Execute **make** from the terminal to compile allegro. If this process completes with no errors, type **sudo make install** to complete installation.

Once Allegro is installed, the game can be compiled using the “CMakeLists.txt” file in the root directory. To install using terminal, navigate to [...]/Escape-the-clocktower-alpha/ then run **cmake -Bbuild -H.**

cmake will create a /build/ directory inside /Escape-the-clocktower-alpha/ containing a makefile. Using Terminal, navigate to /build/, then execute **make**. This will create **etc.exec** in the /build/ directory which can then run using Terminal or Finder.

Overview

Escape the Clocktower is a first-person dungeon crawler/escape the room game, with the theme of escaping the Otago University Clocktower.
Our coding standards are Allman layout, self-explanatory variable and method names, sensible structuring.

Data structures

The two main data structures we've created are the *map* and the *player*.

Map

The *map* is a two dimensional arrays of *MapTile* structs. Right now there's exactly one; when we add more levels to our 'tower', we might make them separate arrays or replace its contents when the player changes levels.
A MapTile struct contains pointers **N, *E, *S, *W* to the images that are displayed when the player is facing north, east, south and west. The same images can be used by multiple tiles, and can be *NULL*.
A MapTile struct contains a boolean value *passable* to indicate whether the player can enter that tile or not. Impassable tiles will usually have null image pointers, but when we implement unlocking new areas we intend to initialise the tile with the images it will use, and only alter the passable value from false to true.
We intend to implement items and NPCs as a separate data type that will contain the necessary information within itself; the only change to the map tiles is expected to be adding a single pointer to the object contained in that tile, or nothing.

Player

The *player* structure contains the position of the player in the world: two integers for the x and y coordinates on the map, and a char that should contain 'N', 'E', 'S' or 'W' to indicate the direction the player is facing.
We intend to implement an inventory system and flags; those are expected to be part of this struct.

Keys enum

MYKEYS enumerates the codes that are later used to distinguish pressing the up, down, left and right arrow keys or the spacebar.

Other global variables

done tracks whether the game is finished. It is a boolean value, initialised to false when the game begins, that is set to true when the user presses escape or closes the game window.
redraw tracks whether the graphics need to be updated. It is a boolean value, initialised to true when the **gameloop** begins. It is set to false if nothing happens that requires a graphics update; is confirmed as true if
key tracks what key has been pressed. It is a pointer to an array of boolean values, initialised to false when the array is declared. The entries in the array are to be accessed using the *MYKEYS* enum for indices.
first distinguishes the first keypress from the user. It is a boolean value, initialised to true when the variable is declared. While it is true, all keypresses other than 'space' are ignored; when the first 'space' is entered to move past the title screen, this variable is set to false.

p is the pointer to the player struct. It is initialised by the method **init_player**.

Methods

abort_game ends the game in case of error. It takes an error message as a parameter, prints that message, and exits the program.

Main

The main method has the standard C parameters. It calls five methods, none of which take parameters or return values.

init sets up Allegro for the session. It initialises the assets from Allegro: the library itself, keyboard, timer, image display, a bitmap of our title screen and event queue. It checks that these have succeeded, and calls abort_game if not.

init_map creates the map. Currently this is hardcoded to set the images and passability of each map tile; we intend to replace this with reading in data from an external file later.

init_player initialises the pointer p. It calls player_constructor to create the player struct, and sets the global variable p to point to that struct.

game_loop runs the game. While the game is not finished [*done*] it uses the Allegro event queue to watch for input. Escape or closing the game window end the game. Otherwise, it calls get_user_input to handle user input. Then it calls update_graphics to redraw the game display if necessary.

shutdown closes the game. It calls functions to destroy the Allegro objects, and the player struct.

Player methods

player_get_position_x,
player_get_position_y and
player_get_direction_facing take the player struct as a parameter, and return the x coordinate, y coordinate and current direction.

player_set_position takes the player struct and two integer coordinates as parameters. Sets the player's new coordinates to the input coordinates.
player_set_direction takes the player struct and a single character direction as parameters. Sets the player's new direction to the input direction.

player_constructor makes the player at the starting point of the map. It allocates memory for the player struct, initialises the position to 0, 0, facing N, and returns a pointer to this struct.
player_destruct takes player struct as a parameter, frees it. Currently this takes one command but if the struct is extended to include other allocations, the corresponding frees will go here.

Gameplay methods

get_user_input takes the keypress event as a parameter and checks what key was pressed. It has no return value. The first keypress must be 'space'. 'Up', 'left' and 'right' cause the relevant value in the keys[] array to be set to true. 'Down' is not currently used, but might be later.

If up arrow key: update the player location by +1 if facing North, or -1 if facing South, after checking is that tile is passable. Moving East or West is currently disabled. If movement is successful, it requires graphics update.
If left or right arrow key: update the direction that the player is facing, counterclockwise or clockwise respectively. Requires graphics update.
All other keypresses print an error message and do not require the graphics to update.

graphics_update changes the graphics to reflect the player's actions. It is called by the gameplay loop after get_user_input, and depends on that method, but through the global variables *first* and *keys[]*: it does not take parameters or return values.
If 'space' was pressed and it's the first keypress, or if it's *not* the first keypress and 'up', 'left' or 'right' was pressed, then the method checks the current location and direction of the player and feeds the relevant image into Allegro's bitmap and display functions to make it show up.