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

Clone/download the repository from https://github.com/cosc345etc/Escape-the-Clocktower.
Open the project file **Escape-the-Clocktower/Application/EtC_Beta/EtC_Beta.xcodeproj/project.pbxproj** in XCode. Press XCode's build-and-run button.

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. There is exactly one; different 'levels' are simply separated by impassable areas, with teleportation from the exit of one to the entrance of the next. It is currently nine by eight tiles.
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, in the format of an *SDL_Surface*. The same images can be used by multiple tiles, and can be *NULL*.
A MapTile struct contains four boolean values *passable_N S, E, W* to indicate whether the player can enter that tile from that direction or not.
There are three variables associated with interacting with a tile: the boolean *is_interactive*, that indicates whether pressing the interact button has an effect in the tile; the character *dir_need_to_face* that specifies the direction the player must be facing to interact with the tile; and a pointer to the SDL_Surface *interact_image* which is displayed after the interaction has occurred.

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, most likely as a bitfield in 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.

Width and Height

These global constants specify that the main game window is a square 480 pixels across.

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 and *second* distinguish the first two keypresses from the user. They are boolean values, 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, *first* is set to false; when the second 'space' is entered to move past the introductory text, *second* is set to false.

columns and *rows* are integers used in the **load_map_file** method.

window, renderer, loadImage and texture are pointers to SDL data types that are used to display the images.

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 SDL for the session. It initialises the assets from SDL: the library itself, window, renderer, title image and display. 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 SDL WaitEvent method 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 SDL texture, surface, renderer and window, 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 7, 1, 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 two keypresses 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 in the direction that the player is facing, after checking is that tile to be entered is passable in that direction. 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.
If 'space' is pressed, the player may be teleported to the next level if in the exit of the first level, or else will check for interactions. Requires graphics update.
Pressing 'escape' will exit the game.
All other keypresses print an error message and do not require the graphics to update.

graphics_show_direction_facing takes and returns nothing. It checks the direction that the player is facing, and sets *texture* to the corresponding image.
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. It does reset the key[] array to be ready for the next keypress.
If 'space' was pressed and it's the first or second keypress, the game moves from the title screen to the introduction screen or onto the normal view respectively.
If 'space' is pressed later, the method checks if the player's current tile and direction faced can be interacted with, fetches the new image, sets the tile to be uninteractable, and sets the

relevant direction image for the current tile to the new image. If it isn't interactable, the current image is shown.
If 'up', 'left' or 'right' was pressed, the method simply resets the keys and shows the direction now being faced.

Map Methods

check_true_or_false is used to check passability.
load_map_file loads the external map file. It takes the file name as input, gets each line in turn, reads it, and closes the file when done.
read_line_to_map reads the map data. It reads in 10 items per line: the four images for the directions the player is facing, the four passable conditions for those directions, the image or lack thereof to be displayed if the square is interacted with, and the direction that the player must be facing to interact successfully.
init_map initialises the map.