COSC 345

"Small is Beautiful"

Assignment 2: Alpha Release

Developer Documentation

Group: CodeBound

Project Title: Devolution

Marcus Anderson
(ID: 7049664)

Surayo Akramkhanova
(ID: 7497059)

Grace Auckram
(ID: 7659739)

Josef Bode
(ID: 5636437)

How to build and run the project:

1. Go to the GitHub and download the file tagged with “Alpha”.
2. Open the Xcode project file.
3. Create a new scheme in Xcode for the assignment.
4. Build and run the program in Xcode.

Playing the game:

Currently we have only implemented keyboard events, mouse functionality will be implemented in the future. Use the up and down arrow keys to control the ‘selector’ and the return key to select any option the selector is positioned over. The escape key can be used to return to the title screen when playing the game, or to exit the game you are currently at the title screen.

What the code does:

At the moment you are able to switch between different storylines depending on what choices you make. For the alpha release we have two options available, a text-based adventure in the console or a GUI representation. At the moment running the game through the console allows the user to explore the story up to the point we are currently writing it. There are currently several ‘random’ bugs where seemingly random text is printed. There is also a memory issue occuring which we haven’t been able to pin down but we will fix it for the beta release.

A description of each source file and their associated functions is provided below:

main.c

The application file for where the program first runs from. It creates the first instance of the GUI window where the program will be running from.

Text handling algorithms:

mylib.c

mylib.c holds two memory allocating methods emalloc and erealloc. These two methods are used for calling malloc and realloc as well as checking that the memory was allocated correctly.

fileLoader.c

fileLoader.c has three main functions, it is able to get the location of the current working directory as well as opening and closing a selected file.

textManipulation.c

- The setFile method loads all of text from the file into memory and returns a pointer to the loaded text
- The storeBrackets method calls the removeBrackets method in which the text is cleaned and all of the brackets are removed and replaced representing the text in the correct manner. For example at the moment where [NAME] is in the file it is replaced with “Greg” and any other important brackets will be replaced by this.
- The storeBrackets method also sets up all of the choices that user can select into structs with a string pointer to the next file as well as the text for each choice. This makes it so that the choices can be retrieved individually and they will have strings pointing to the next file.
- The storeBrackets method also contains a few get methods which can take a users input and return the specified story branch that the user has selected

Graphical User Interface:

We are using SDL2 to implement GUI elements. (https://www.libsdl.org/index.php)
The window size is fixed at 1366 x 768 pixels wide as this is a common screen resolution and allows for good presentation of the GUI elements (i.e. text is readable).

I will go through each C file relevant to the GUI and explain what each function does.

graphics.c

(create window, load game, render GUI elements in a loop, check for keyboard events)

createWindow

This function is called by main when the user wants to create a GUI version of the game.
Any libraries are initialised, including SDL2 and SDL2_ttf. SDL2 is for low level GUI elements (we used it for video but will most likely be using images and sound in later releases). SDL2_ttf is for displaying text through the core SDL2 library.
Using SDL, window creation uses straightforward parameters to set the title, position and size of the window (and any flags - I used OpenGL by default). The window is then passed into a renderer which is used to draw any elements onto the screen.
The functions for loading the game, event processing and rendering the display are all utilised in the createWindow function to ensure the game runs properly.
When the user carries out an action to close the game, thus exiting the event processing loop, we need to ensure that we destroy any GUI elements that we previously declared. This is all done at the end of the function where we close the fonts, destroy the window and renderer, and quit the libraries that we used.

loadGame
I use this function to load in any required resources and set the game to begin with the title screen. I currently only load in font elements as I require them for all the text used throughout the program. In the future I will use images which will need to be loaded in via this function. I also set the "screenCenter" variables to the x and y coordinates of the center of the window. By doing this I can position GUI elements relative to the screen center.

processEvents
Used for checking for any events, this is currently only for user input. The function is constantly executing a loop to check whether a key has been pressed. Used a switch statement to track different key pressed. Using this function, we can move the selector and use it to select options in the title screen or choices in the game. We also use this function to allow the user to exit the game without any instability.

doRender
Renders the display according to the status of the game (i.e. if the status is set to STATUS_STATE_TITLE then draw the title screen). This function is constantly executed in a loop to keep the relevant screen (title or game) displayed.

titleScreen.c

(draws the GUI elements relevant to the titleScreen)

init_title_screen
Creates any text to be rendered on the title screen by creating a surface then rendering the text onto that surface. The surface is then used to create a texture that is later drawn by the renderer. For the title screen, I created 6 individual textures to be displayed.
It is also important to move the selector into the right spot and set the necessary dimensions (it is initially located over the new game option).

draw_title_screen
Draws every element and displays them on the title screen.

shutdown_title_screen
Destroys any textures used.

gameScreen.c

(draws the GUI elements relevant to the gameScreen)

init_game_screen
Moves the selector over the first choice and sets the necessary dimensions.

draw_game_screen
Draws every element and displays them on the game screen. This just draws the shapes, the text for the story and choices are drawn in displayText.c so we call the relevant function after the shapes have been drawn.

shutdown_game_screen
Destroys any textures used (calls the shutdown_display_text function to destroy textures used).

displayText.c

(draws the text used in the gameScreen)

init_display_text
Grabs the text (story and choices) from the specified text files and uses these to create textures.

draw_display_text
Draws the story text and each of the choices on their corresponding shapes.

changeScenario
Used to further the story by changing the scenario status.

shutdown_display_text
Destroys the textures used.

It is also worth going over the graphics header file as it contains important structs and useful definitions:

graphics.h

I used several definitions throughout the program as this made it easier to keep track of where the selector is, which scenario is currently active, or whether the game state is set to title screen or gameplay. I defined two structs (Selector and GameState) in this header file as I use them extensively throughout the GUI program. The Selector struct contains the x, y, width and height values which change between the title and game screens. The GameState struct contains every variable shared across the GUI which is why almost every function is parameterised by GameState.