

```
1  #ifndef MENU_H
2  #define MENU_H
3  #include "Menu.h"
4  #endif
5  #include <stdio.h>
6  #include "Window.h"
7  #include "string"
8  #include "array"
9  #include "Mixer.h"
10
11 Menu::Menu() {
12     running = false;
13     firstMenuOptionPickedFlag = false;
14     curr_cursor_pos = 1;
15     distance_between_menu_option = 50;
16     menu_items.clear();
17     unused_menu_items.clear();
18 };
19
20 void Menu::render() {
21     //Set Renderer
22     SDL_Renderer* renderer = Window::getInstance()->getRenderer();
23
24     //Clear Screen, Set Background Color
25     SDL_RenderClear(renderer);
26     SDL_SetRenderDrawColor(renderer, 0x11, 0x11, 0x11, 0xFF);
27
28     //render menu items
29     for (int i = 0; i < menu_items.size(); i++) {
30         menu_items.at(i)->render();
31     }
32
33     //Present Renderer
34     SDL_RenderPresent(renderer);
35 }
36
37 void Menu::tick() {
38     for (int i = 0; i < menu_items.size(); i++) {
39         menu_items.at(i)->tick();
40     }
41 };
42
43 void Menu::moveCursor(std::string input) {
44     if (menu_items.size() < 2) {
45         printf("Currsr could not be placed\n");
46         return;
47     }
48
49     //Find Cursor
50     int cursor = menu_items.size();
51     int last_el = menu_items.size()-2;
52     for (int i = 0; i < menu_items.size(); i++) {
53         if (menu_items.at(i)->getType().compare("CURSOR") == 0) {
```

```
54         cursor = i;
55     }
56     if (menu_items.at(i)->getType().compare("CURSOR") == 1) {
57         last_el = i;
58     }
59 }
60
61 //Check if
62 if (input.compare("UP") == 0) {
63     //If Cursor is not already on the first menu option element
64     if (curr_cursor_pos > 1) {
65         //set cursor one more up
66         menu_items.at(cursor)->setY(menu_items.at(cursor)->getY() -  ↗
            menu_items.at(1)->getH() - distance_between_menu_option);
67         curr_cursor_pos -= 1;
68     }
69     else if (curr_cursor_pos == 1){
70         //set cursor to lowest element
71         menu_items.at(cursor)->setY(menu_items.at(last_el)->getY() +  ↗
            (menu_items.at(last_el)->getH() / 4));
72         curr_cursor_pos = menu_items.size() - 4;
73     }
74 }
75 else if (input.compare("DOWN") == 0) {
76     //If Cursor is not at the last element
77     if (curr_cursor_pos < (menu_items.size() - 4)) {
78         //set cursor one more up
79         menu_items.at(cursor)->setY(menu_items.at(cursor)->getY() +  ↗
            menu_items.at(1)->getH() + distance_between_menu_option);
80         curr_cursor_pos += 1;
81     }
82     else if (curr_cursor_pos >= (menu_items.size() - 4)) {
83         //set cursor to first element
84         menu_items.at(cursor)->setY(menu_items.at(1)->getY() +  ↗
            (menu_items.at(1)->getH() / 4));
85         curr_cursor_pos = 1;
86     }
87 }
88
89 //Play Sound
90 Mixer::getInstance()->play(Mixer::SWORDDRAWN2);
91 };
92
93 void Menu::input() {
94     SDL_Event e;
95
96     //Exit if quit pressed
97     while (SDL_PollEvent(&e)) {
98
99         if (e.type == SDL_QUIT) {
100             running = false;
101         }
102     }
```

```
103     if (e.type == SDL_KEYDOWN) {
104         if (e.key.keysym.sym == SDLK_RETURN) {
105             switch (curr_cursor_pos)
106             {
107                 case 1:
108                     running = false;
109                     firstMenuOptionPickedFlag = true;
110                     Mixer::getInstance()->playMusic(Mixer::FIGHTSONG);
111                     break;
112                 case 2:
113                     running = false;
114                     firstMenuOptionPickedFlag = false;
115                     break;
116                 default:
117                     break;
118             }
119         }
120
121         if (e.key.keysym.sym == SDLK_ESCAPE) {
122             running = false;
123             firstMenuOptionPickedFlag = true;
124             Mixer::getInstance()->playMusic(Mixer::FIGHTSONG);
125         }
126
127         if (e.key.keysym.sym == SDLK_DOWN) {
128             moveCursor("DOWN");
129         }
130
131         if (e.key.keysym.sym == SDLK_UP) {
132             moveCursor("UP");
133         }
134     }
135 }
136 };
137
138 bool Menu::show(int flag) { // "false" for [exit], "true" for [start game] ↗
139     or continue or restart]
140     //Check what menu item shall be shown
141     if ((flag >= 0) && (flag <= 2))
142         {changeFirstMenuItemTo(flag);}
143     else
144         {changeFirstMenuItemTo(CONTINUE);}
145
146     //Set flags
147     firstMenuOptionPickedFlag = false;
148     running = true;
149
150     //Start menu loop
151     while (running) {
152         input();
153         tick();
154         render();
155     }
```

```
155
156     return firstMenuOptionPickedFlag;
157 };
158
159 bool Menu::loadMedia() {
160     bool success = true;
161
162     for (int i = 0; i < menu_items.size(); i++) {
163         if (!menu_items.at(i)->loadMedia()) {
164             printf("Failed to load menu item Number: %d\n", i+1);
165             success = false;
166             break;
167         }
168
169         //PLACE MENU ITEMS
170         //Set y-coordinate of menu options
171         if (i == 1) {
172             menu_items.at(i)->setY(menu_items.at(0)->getH() + 2 *      ↗
173                 distance_between_menu_option);
174
175         }
176         else if (i > 1) {
177             menu_items.at(i)->setY(menu_items.at(i - 1)->getY() +      ↗
178                 menu_items.at(i - 1)->getH() + distance_between_menu_option);
179         }
180
181         //Set x-coordinate
182         if (i <= 1) {
183             //Place every menu option in the middle of the screen width
184             menu_items.at(i)->setX((Window::getInstance()->getWindowSizeW      ↗
185                 () - menu_items.at(i)->getW()) / 2);
186         }
187         else if (i > 1) {
188             //Place every menu option after first one to the same x -      ↗
189                 coordinate as the first menu option
190             menu_items.at(i)->setX(menu_items.at(i - 1)->getX());
191         }
192
193         //Place Cursor - if .compare(...) == 0 it means the strings are      ↗
194             equal
195         if ((menu_items.at(i)->getType().compare("CURSOR") == 0)) {
196             menu_items.at(i)->setY(menu_items.at(1)->getY() +      ↗
197                 (menu_items.at(1)->getH() / 4));
198             menu_items.at(i)->setX(menu_items.at(1)->getX() -      ↗
199                 menu_items.at(i)->getW() - 50);
200             curr_cursor_pos = 1;
201         }
202
203         //Place Control textures
204         if ((menu_items.at(i)->getType().compare("CONTROLSTX") == 0)) {
205             menu_items.at(i)->setX(((Window::getInstance()->getWindowSizeW      ↗
206                 () / 2) - menu_items.at(i)->getW())/2);
207             menu_items.at(i)->setY(menu_items.at(2)->getY() +      ↗
208                 menu_items.at(2)->getH());
209             if (i >= menu_items.size() - 2) {
```

```
199         int tmp = (Window::getInstance()->getWindowSizeW() / 2) +  
                menu_items.at(i)->getW();  
200         tmp = (Window::getInstance()->getWindowSizeW() - tmp)/2;  
201         tmp += Window::getInstance()->getWindowSizeW() / 2;  
202         menu_items.at(i)->setX(tmp);  
203     }  
204 }  
205 }  
206  
207 //Load later used menu items  
208 for (int i = 0; i < unused_menu_items.size(); i++) {  
209     if (!unused_menu_items.at(i)->loadMedia()) {  
210         printf("Failed to load menu item.\n");  
211         success = false;  
212     }  
213     else {  
214         //Set Coordinates for later used menu items  
215         unused_menu_items.at(i)->setY(menu_items.at(1)->getY());  
216         unused_menu_items.at(i)->setX(menu_items.at(1)->getX());  
217     }  
218 }  
219 return success;  
220 };  
221  
222 bool Menu::init() {  
223     bool success = true;  
224  
225     //Set up values for Menu Items  
226     std::string fontpath1 = "assets/fonts/PlayfairDisplay-  
        BlackItalic.ttf";  
227     std::string fontpath2 = "assets/fonts/PlayfairDisplay-Italic.ttf";  
228     MenuTexture* menutx = NULL;  
229     SDL_Color color1 = { 213, 0, 28, 255 };  
230     SDL_Color color2 = { 255, 184, 81, 255 };  
231     std::array<std::string, 3> titles = { "Big City Knights", "Start  
        Game", "Exit" };  
232  
233     for (int i = 0; i < titles.size() + 1; i++) {  
234         if (i == 0) {  
235             menu_items.push_back(new MenuTexture(titles[i],  
                fontpath1.c_str(), 112/*112*/, color1));  
236         }  
237         else if(i < titles.size()){  
238             menu_items.push_back(new MenuTexture(titles[i],  
                fontpath2.c_str(), 48, color2));  
239         }  
240         else {  
241             menu_items.push_back(new MenuControlsTexture("assets/  
                sprite_sheets/menu/MENULEFTPLAYER.png"));  
242             menu_items.push_back(new MenuControlsTexture("assets/  
                sprite_sheets/menu/MENURIGHTPLAYER.png"));  
243             menu_items.push_back(new MenuCursor());  
244         }
```

```
245     }
246
247     //Menu Items that are used later
248     unused_menu_items.push_back(new MenuTexture("Resume", fontpath2.c_str  ↗
        (), 48, color2));
249     unused_menu_items.push_back(new MenuTexture("Restart", fontpath2.c_str  ↗
        (), 48, color2));
250     unused_menu_items.push_back(new MenuTexture("Start Game",           ↗
        fontpath2.c_str(), 48, color2));
251
252     return success;
253 };
254
255 void Menu::changeFirstMenuItemTo(int x) {
256     menu_items.erase(menu_items.begin() + 1);
257     menu_items.emplace(menu_items.begin() + 1, unused_menu_items.at(x));
258 };
259
260 void Menu::close() {
261     //Free menu items
262     for (int i = 0; i < menu_items.size(); i++) {
263         menu_items.at(i)->free();
264     }
265     menu_items.clear();
266
267     //Free unused menu items
268     for (int i = 0; i < unused_menu_items.size(); i++) {
269         unused_menu_items.at(i)->free();
270     }
271     unused_menu_items.clear();
272 };
```