```cpp
#include "Player.h"
#include "SDL_image.h"
#include "Window.h"
#include "math.h"
#include "Collider.h"
#include "ObservableCollisionDetection.h"
#include "Mixer.h"

Player::Player(bool headLeft){
    //Getting the Window Sizes
    int window_size_h = Window::getInstance()->getWindowSizeH();
    int window_size_w = Window::getInstance()->getWindowSizeW();

    //Setting up Size, Position, States, Direction in which the Player heads, ticked (same as PlayerLeft)
    appearance.x = 50;
    appearance.y = window_size_h - (window_size_h / 4);
    appearance.w = 128;
    appearance.h = 128;
    this->headLeft = headLeft;
    for (int i = 0; i < spritesheets.size(); i++) {
        spritesheets[i] = NULL;
    }
    blocking = false;
    ticked = 0;
    heightAboveTheGround = 0;
    curr_state[0] = 0;
    curr_state[1] = IDLE;
    max_sprites = { 4,8,3,4,5,5,5 };
    while (!height_stack.empty()) {
        height_stack.pop();
    }
};

bool Player::loadMedia() {
    bool success = true;

    std::string sources[9];
    if (headLeft == true) {
        sources[0] = { "assets/sprite_sheets/player/Right_Player/sheet_hero_idle.png" };
        sources[1] = { "assets/sprite_sheets/player/Right_Player/sheet_hero_block.png" };
        sources[2] = { "assets/sprite_sheets/player/Right_Player/sheet_hero_walk.png" };
        sources[3] = { "assets/sprite_sheets/player/Right_Player/sheet_hero_hurt.png" };
        sources[4] = { "assets/sprite_sheets/player/Right_Player/sheet_hero_jump.png" };
        sources[5] = { "assets/sprite_sheets/player/Right_Player/sheet_hero_stab.png" };
        sources[6] = { "assets/sprite_sheets/player/Right_Player/sheet_hero_throw_bottle.png" };
```

```
46      }
47      else {
48          sources[0] = { "assets/sprite_sheets/player/Left_Player/
              sheet_hero_idle.png" };
49          sources[1] = { "assets/sprite_sheets/player/Left_Player/
              sheet_hero_block.png" };
50          sources[2] = { "assets/sprite_sheets/player/Left_Player/
              sheet_hero_walk.png" };
51          sources[3] = { "assets/sprite_sheets/player/Left_Player/
              sheet_hero_hurt.png" };
52          sources[4] = { "assets/sprite_sheets/player/Left_Player/
              sheet_hero_jump.png" };
53          sources[5] = { "assets/sprite_sheets/player/Left_Player/
              sheet_hero_stab.png" };
54          sources[6] = { "assets/sprite_sheets/player/Left_Player/
              sheet_hero_throw_bottle.png" };
55      }
56
57      //Load Spritesheets
58      for (int i = 0; i < spritesheets.size(); i++) {
59          spritesheets[i] = loadTexture(sources[i]);
60          if (spritesheets[i] == NULL) {
61              printf("Failed to create texture. SDL Error: %s\n",
                    SDL_GetError());
62              success = false;
63          }
64      }
65
66      //Create Healthbar
67      healthbar = new Healthbar(!headLeft);
68      if (healthbar == NULL) {
69          printf("Failed to create Healthbar\n");
70          success = false;
71      }
72
73      return success;
74  };
75
76  void Player::renderUnflipped() {
77      //Set up Clip Rectangle
78      SDL_Rect clip_rect;
79      clip_rect.x = curr_state[0] * 64;
80      clip_rect.y = 0;
81      clip_rect.w = 64;
82      clip_rect.h = 64;
83
84      //Render
85      SDL_RenderCopy(Window::getInstance()->getRenderer(),
86          spritesheets[curr_state[1]], &clip_rect, &appearance);
87  };
88
89  void Player::renderFlipped() {
90      //Set up Clip Rectangle
```

```
 91      SDL_Rect clip_rect;
 92      clip_rect.x = curr_state[0] * 64;
 93      clip_rect.y = 0;
 94      clip_rect.w = 64;
 95      clip_rect.h = 64;
 96
 97      //Render
 98      SDL_RenderCopyEx(Window::getInstance()->getRenderer(), spritesheets
           [curr_state[1]],
 99          &clip_rect, &appearance, 0.0, NULL, SDL_FLIP_HORIZONTAL);
100  };
101
102  void Player::renderColliders() {
103      //Saving the old rendercolor
104      SDL_Color old_color;
105      SDL_GetRenderDrawColor(Window::getInstance()->getRenderer(),
           &old_color.r, &old_color.g, &old_color.b, &old_color.a);
106
107      //Set player color
108      SDL_Color color;
109      color.r = 0x00;
110      color.g = 0xFF;
111      color.b = 0x00;
112      color.a = 0x70;
113
114      SDL_SetRenderDrawColor(Window::getInstance()->getRenderer(), color.r,
           color.g, color.b, color.a);
115
116      //Get colliders
117      std::vector<SDL_Rect> colliders = getColliders();
118
119      //Render
120      for (int i = 0; i < colliders.size(); i++) {
121          if (i == 1) {
122              //Set Sword Color
123              color.r = 0xFF;
124              color.g = 0x00;
125              SDL_SetRenderDrawColor(Window::getInstance()->getRenderer(),
                   color.r, color.g, color.b, color.a);
126          }
127          SDL_RenderFillRect(Window::getInstance()->getRenderer(), &
               (colliders.at(i)));
128      }
129
130      //Reset the old rendercolor
131      SDL_SetRenderDrawColor(Window::getInstance()->getRenderer(),
           old_color.r, old_color.g, old_color.b, old_color.a);
132  };
133
134  void Player::render() {
135      //Render Player
136      if (headLeft) {
137          renderFlipped();
```

```
138     }
139     else {
140         renderUnflipped();
141     }
142 
143     //Render Healthbar
144     healthbar->render();
145 
146     //Render Colliders from Player
147     if (false) { renderColliders(); }
148 };
149 
150 void Player::tick() {   /*METHOD GETS OVERWRITTEN IN SUBCLASSES*/ };
151 
152 void Player::moveX(bool caused_by_collision, int distance) {
153     //If no hurt and
154     if (caused_by_collision == false) {
155         if (headLeft == true) {
156             //Left Border/Edge of the Screen
157             if (appearance.x > -32) {
158                 int i = 0;
159                 while ((i < distance) && (appearance.x > -32)) {
160                     appearance.x -= 1;
161                     i++;
162                 }
163             }
164         }
165         else {
166             //Right Border/Edge of the Screen
167             if ((appearance.x + appearance.w) < Window::getInstance()-
                  >getWindowSizeW() + 32) {
168                 int i = 0;
169                 while ((i < distance) && ((appearance.x + appearance.w +
                      1) < Window::getInstance()->getWindowSizeW() + 32)) {
170                     appearance.x += 1;
171                     i++;
172                 }
173             }
174         }
175     }
176     else{
177         if (collision_direction.compare("RIGHT") == 0) {
178             //Left Border/Edge of the Screen
179             if (appearance.x > -32) {
180                 int i = 0;
181                 while ((i < distance) && (appearance.x > -32)) {
182                     appearance.x -= 1;
183                     i++;
184                 }
185             }
186         }
187         else if(collision_direction.compare("LEFT") == 0){
188             //Right Border/Edge of the Screen
```

```
189             if ((appearance.x + appearance.w) < Window::getInstance()->getWindowSizeW() + 32) {
190                 int i = 0;
191                 while ((i < distance) && ((appearance.x + appearance.w + 1) < Window::getInstance()->getWindowSizeW() + 32)) {
192                     appearance.x += 1;
193                     i++;
194                 }
195             }
196         }
197     }
198 };
199
200 void Player::changeStateTo(int state) {
201     //Making sure that HURT and JUMP can not be interrupted
202     if ((curr_state[1] != HURT) && (curr_state[1] != JUMP)) {
203         if ((state == JUMP) && (heightAboveTheGround > 0)) {}
204         /*It has to be possible to stab/throw a bottle while walking,
205             therefore WALK can not interrupt STAB/THROWBOTTLE*/
206         else if ((state == WALK) && ((curr_state[1] == STAB) || (curr_state[1] == THROWBOTTLE))) {}
207         else {
208             curr_state[0] = 0;
209             curr_state[1] = state;
210             ticked = 0;
211         }
212     }
213 };
214
215 void Player::checkInput() {/*Gets overwritten in subclasses*/};
216
217 Bottle* Player::spawnBottle() {
218     Bottle* bottle = NULL;
219
220     //Only throw one bottle while throwing
221     if ((curr_state[1] == THROWBOTTLE) && (curr_state[0] == 2) && (ticked == 0)) {
222         if (headLeft == false) {
223             bottle = new Bottle(appearance.x + (appearance.w / 2), appearance.y + appearance.h - 18, headLeft);
224             bottle->loadMedia();
225         }
226         else {
227             bottle = new Bottle(appearance.x - 64 + (appearance.w / 2), appearance.y + appearance.h - 18, headLeft);
228             bottle->loadMedia();
229         }
230     }
231
232     return bottle;
233 };
234
235 void Player::update(int collided_with, int own_collider, SDL_Rect rec) {
```

```
236     //If blocking or Hurt nothing should happen
237     if ((curr_state[1] == BLOCK) || (curr_state[1] == HURT)) {
238         return;
239     }
240
241     //Find out from which side the hit came from and setting up the
          collision direction for it
242     collision_direction = "NONE";
243     std::vector<SDL_Rect> colliders = getColliders();
244
245     //colliders.at(0) equals this players BODY rectangle
246     //rec is the rectangle the player collided with
247     if (rec.x + (rec.w/2) >= (colliders.at(0).x + (colliders.at(0).w/2)))
        {
248         collision_direction = "RIGHT";
249     }
250     else {
251         collision_direction = "LEFT";
252     }
253
254     //React
255     switch (collided_with)
256     {
257     case BODY:
258         if (own_collider == BODY) {
259             moveX(true, 4);
260         }
261         break;
262     case BOTTLE:
263         changeStateTo(HURT);
264         healthbar->takeDamage(50);
265         Mixer::getInstance()->play(Mixer::HURT);
266         break;
267     case SWORD:
268         if (own_collider != SWORD) {
269             changeStateTo(HURT);
270             healthbar->takeDamage(200);
271             Mixer::getInstance()->play(Mixer::HURT);
272         }
273         else {
274             Mixer::getInstance()->play(Mixer::SWORDDRAWN1);
275         }
276         break;
277     default:
278         break;
279     }
280 };
281
282 int Player::getColliderType(int index_in_vector) {
283     int type = -1;
284
285     if (index_in_vector == 0) {
286         type = Player::BODY;
```

```
287     }
288     else if ((index_in_vector == 1) && (curr_state[1] ==
          Player::THROWBOTTLE)) {
289         type = Player::BOTTLE;
290     }
291     else if (index_in_vector == 1) {
292         type = Player::SWORD;
293     }
294
295     return type;
296 };
297
298 std::vector<SDL_Rect> Player::getColliders() {
299     Collider colliderInst;
300     std::vector<SDL_Rect> colliders;
301
302     //Green Player Rectangle
303     colliders.push_back(colliderInst.getColliderForPlayer(curr_state[1],
          curr_state[0], headLeft, appearance));
304
305     //If Red Sword Rectangle exists
306     SDL_Rect sword_rec = colliderInst.getColliderForPlayerSword(curr_state
          [1], curr_state[0], headLeft, appearance);
307     if ((sword_rec.x == 0) && (sword_rec.y == 0) && (sword_rec.w == 0) &&
          (sword_rec.h == 0)) {}
308     else {
309         colliders.push_back(sword_rec);
310     }
311
312     return colliders;
313 };
314
315 int Player::getState() {
316     return curr_state[1];
317 };
318
319 std::string Player::isDead() {
320     std::string isDead = "NOTDEAD";
321
322     if (healthbar->isEmpty()) {
323         isDead = "DEADPLAYER";
324     }
325
326     return isDead;
327 };
328
329 std::string Player::getType() {
330     return "PLAYER";
331 };
332
333 void Player::restart() {/*Gets overwritten in sub classes*/};
334
335 void Player::close() {
```

```
336     //Destroy Healthbar
337     healthbar->~Healthbar();
338     healthbar = NULL;
339 
340     //Destroy Textures
341     for (int i = 0; i < spritesheets.size(); i++) {
342         if (spritesheets[i] != NULL) {
343             SDL_DestroyTexture(spritesheets[i]);
344             spritesheets[i] = NULL;
345         }
346     }
347 
348 };
349 
350 Player::~Player() {
351     close();
352 }
```