```
    1: // Copyright 2023 Thomas O'Connor
    2: #include "Checkers.hpp"
    3:
    4: // Getter
    5: bool Checkers::isWon(void) {
    6:     if (setWinTrue) return 1;
    7:     bool r = 0, b = 0;
    8:     for (int i = 0; i < BOARD_DIMENSIONS; i++) {
    9:         for (int j = 0; j < BOARD_DIMENSIONS; j++) {
   10:             if (currentGameState[i][j] == 'r' ||
   11:                 currentGameState[i][j] == 'R')
   12:                 r = 1;
   13:             if (currentGameState[i][j] == 'b' ||
   14:                 currentGameState[i][j] == 'B')
   15:                 b = 1;
   16:             if ((r && b) || stillPlaying) return 0;
   17:         }
   18:     }
   19:     return 1;
   20: }
   21:
   22: // Getter (ONLY CALL AFTER ISWON IS CONFIRMED)
   23: bool Checkers::getWinner(void) {
   24:     for (int i = 0; i < BOARD_DIMENSIONS; i++) {
   25:         for (int j = 0; j < BOARD_DIMENSIONS; j++) {
   26:             // if a red piece exists, red wins
   27:             if (currentGameState[i][j] == 'r' ||
   28:                 currentGameState[i][j] == 'R')
   29:                 return 1;
   30:         }
   31:     }
   32:     // else black wins
   33:     return 0;
   34: }
   35:
   36: // Interactor
   37: void Checkers::selectPiece(sf::Vector2i mouseLocation) {
   38:     // calculate the coordiate pair in relation to the 2D char array
   39:     sf::Vector2i arenaLocation((mouseLocation.x - BOARD_OFFSET) / TILE_SI
ZE,
   40:                 (mouseLocation.y - BOARD_OFFSET) / TILE_SIZE);
   41:     if (playerTurn) {
   42:         // red turn
   43:         if (currentGameState[arenaLocation.y][arenaLocation.x] == 'r') {
   44:             currentGameState[arenaLocation.y][arenaLocation.x] = 'w';
   45:             stillPlaying = 1;
   46:             return;
   47:         }
   48:         if (currentGameState[arenaLocation.y][arenaLocation.x] == 'R') {
   49:             currentGameState[arenaLocation.y][arenaLocation.x] = 'W';
   50:             stillPlaying = 1;
   51:             return;
   52:         }
   53:     } else {
   54:         // black turn
   55:         if (currentGameState[arenaLocation.y][arenaLocation.x] == 'b') {
   56:             currentGameState[arenaLocation.y][arenaLocation.x] = 'w';
   57:             stillPlaying = 1;
   58:             return;
   59:         }
   60:         if (currentGameState[arenaLocation.y][arenaLocation.x] == 'B') {
   61:             currentGameState[arenaLocation.y][arenaLocation.x] = 'W';
   62:             stillPlaying = 1;
   63:             return;
   64:         }
```

```
   65:     }
   66: }
   67:
   68: // Getter
   69: sf::Vector2i Checkers::getSelectedPawn(void) {
   70:     auto row_iter = std::find_if(currentGameState.begin(), currentGameSta
te.end(),
   71:         [](const vector<char>& row) {
   72:         return std::find_if(row.begin(), row.end(), [](char c) {
   73:             return (c == 'w' || c == 'W');
   74:         }) != row.end();
   75:     });
   76:     if (row_iter != currentGameState.end()) {
   77:         auto col_iter = std::find_if(row_iter->begin(), row_iter->end(),
[](char c) {
   78:             return (c == 'w' || c == 'W');
   79:         });
   80:         return sf::Vector2i(std::distance(currentGameState.begin(), row_i
ter),
   81:                 std::distance(row_iter->begin(), col_iter));
   82:     } else {
   83:         return sf::Vector2i();
   84:     }
   85: }
   86:
   87: // Interactor
   88: void Checkers::movePiece(sf::Vector2i mouseLocation) {
   89:     // calculate the coordiate pair in relation to the 2D char array
   90:     sf::Vector2i arenaLocation((mouseLocation.y - BOARD_OFFSET) / TILE_SI
ZE,
   91:                 (mouseLocation.x - BOARD_OFFSET) / TILE_SIZE);
   92:     sf::Vector2i P = getSelectedPawn();
   93:     if (P.x == 0 && P.y == 0) return;
   94:     bool fourD = 0, moved = 0;
   95:     if (currentGameState[P.x][P.y] == 'W') fourD = true;
   96:     if (playerTurn) {
   97:         // red pawn
   98:         if (arenaLocation.x == P.x-1 && arenaLocation.y == P.y-1) {
   99:             if (currentGameState[P.x-1][P.y-1] == 'p') {
  100:                 if (fourD) currentGameState[P.x-1][P.y-1] = 'R';
  101:                 else
  102:                     currentGameState[P.x-1][P.y-1] = 'r';
  103:                 moved = 1;
  104:             }
  105:         }
  106:         if (arenaLocation.x == P.x-1 && arenaLocation.y == P.y+1) {
  107:             if (currentGameState[P.x-1][P.y+1] == 'p') {
  108:                 if (fourD) currentGameState[P.x-1][P.y+1] = 'R';
  109:                 else
  110:                     currentGameState[P.x-1][P.y+1] = 'r';
  111:                 moved = 1;
  112:             }
  113:         }
  114:         if (arenaLocation.x == P.x-2 && arenaLocation.y == P.y-2) {
  115:             if ((currentGameState[P.x-1][P.y-1] == 'b' ||
  116:                 currentGameState[P.x-1][P.y-1] == 'B') &&
  117:                 currentGameState[P.x-2][P.y-2] == 'p') {
  118:                 if (fourD) currentGameState[P.x-2][P.y-2] = 'R';
  119:                 else
  120:                     currentGameState[P.x-2][P.y-2] = 'r';
  121:                 currentGameState[P.x-1][P.y-1] = 'p';
  122:                 moved = 1;
  123:             }
  124:         }
  125:         if (arenaLocation.x == P.x-2 && arenaLocation.y == P.y+2) {
```

```
126:             if ((currentGameState[P.x-1][P.y+1] == 'b' ||
127:                 currentGameState[P.x-1][P.y+1] == 'B') &&
128:                 currentGameState[P.x-2][P.y+2] == 'p') {
129:                 if (fourD) currentGameState[P.x-2][P.y+2] = 'R';
130:                 else
131:                     currentGameState[P.x-2][P.y+2] = 'r';
132:                 currentGameState[P.x-1][P.y+1] = 'p';
133:                 moved = 1;
134:             }
135:         }
136:         if (fourD) {
137:             if (arenaLocation.x == P.x+1 && arenaLocation.y == P.y-1) {
138:                 if (currentGameState[P.x+1][P.y-1] == 'p') {
139:                     currentGameState[P.x+1][P.y-1] = 'R';
140:                     moved = 1;
141:                 }
142:             }
143:             if (arenaLocation.x == P.x+1 && arenaLocation.y == P.y+1) {
144:                 if (currentGameState[P.x+1][P.y+1] == 'p') {
145:                     currentGameState[P.x+1][P.y+1] = 'R';
146:                     moved = 1;
147:                 }
148:             }
149:             if (arenaLocation.x == P.x+2 && arenaLocation.y == P.y-2) {
150:                 if ((currentGameState[P.x+1][P.y-1] == 'b' ||
151:                     currentGameState[P.x+1][P.y-1] == 'B') &&
152:                     currentGameState[P.x+2][P.y-2] == 'p') {
153:                     currentGameState[P.x+2][P.y-2] = 'R';
154:                     currentGameState[P.x+1][P.y-1] = 'p';
155:                     moved = 1;
156:                 }
157:             }
158:             if (arenaLocation.x == P.x+2 && arenaLocation.y == P.y+2) {
159:                 if ((currentGameState[P.x+1][P.y+1] == 'b' ||
160:                     currentGameState[P.x+1][P.y+1] == 'B') &&
161:                     currentGameState[P.x+2][P.y+2] == 'p') {
162:                     currentGameState[P.x+2][P.y+2] = 'R';
163:                     currentGameState[P.x+1][P.y+1] = 'p';
164:                     moved = 1;
165:                 }
166:             }
167:         }
168:     } else {
169:         // black pawn
170:         if (arenaLocation.x == P.x+1 && arenaLocation.y == P.y-1) {
171:             if (currentGameState[P.x+1][P.y-1] == 'p') {
172:                 if (fourD) currentGameState[P.x+1][P.y-1] = 'B';
173:                 else
174:                     currentGameState[P.x+1][P.y-1] = 'b';
175:                 moved = 1;
176:             }
177:         }
178:         if (arenaLocation.x == P.x+1 && arenaLocation.y == P.y+1) {
179:             if (currentGameState[P.x+1][P.y+1] == 'p') {
180:                 if (fourD) currentGameState[P.x+1][P.y+1] = 'B';
181:                 else
182:                     currentGameState[P.x+1][P.y+1] = 'b';
183:                 moved = 1;
184:             }
185:         }
186:         if (arenaLocation.x == P.x+2 && arenaLocation.y == P.y-2) {
187:             if ((currentGameState[P.x+1][P.y-1] == 'r' ||
188:                 currentGameState[P.x+1][P.y-1] == 'R') &&
189:                 currentGameState[P.x+2][P.y-2] == 'p') {
190:                 if (fourD) currentGameState[P.x+2][P.y-2] = 'B';
```

```
191:                 else
192:                     currentGameState[P.x+2][P.y-2] = 'b';
193:                 currentGameState[P.x+1][P.y-1] = 'p';
194:                 moved = 1;
195:             }
196:         }
197:         if (arenaLocation.x == P.x+2 && arenaLocation.y == P.y+2) {
198:             if ((currentGameState[P.x+1][P.y+1] == 'r' ||
199:                 currentGameState[P.x+1][P.y+1] == 'R') &&
200:                 currentGameState[P.x+2][P.y+2] == 'p') {
201:                 if (fourD) currentGameState[P.x+2][P.y+2] = 'B';
202:                 else
203:                     currentGameState[P.x+2][P.y+2] = 'b';
204:                 currentGameState[P.x+1][P.y+1] = 'p';
205:                 moved = 1;
206:             }
207:         }
208:         if (fourD) {
209:             if (arenaLocation.x == P.x-1 && arenaLocation.y == P.y-1) {
210:                 if (currentGameState[P.x-1][P.y-1] == 'p') {
211:                     currentGameState[P.x-1][P.y-1] = 'B';
212:                     moved = 1;
213:                 }
214:             }
215:             if (arenaLocation.x == P.x-1 && arenaLocation.y == P.y+1) {
216:                 if (currentGameState[P.x-1][P.y+1] == 'p') {
217:                     currentGameState[P.x-1][P.y+1] = 'B';
218:                     moved = 1;
219:                 }
220:             }
221:             if (arenaLocation.x == P.x-2 && arenaLocation.y == P.y-2) {
222:                 if ((currentGameState[P.x-1][P.y-1] == 'r' ||
223:                     currentGameState[P.x-1][P.y-1] == 'R') &&
224:                     currentGameState[P.x-2][P.y-2] == 'p') {
225:                     currentGameState[P.x-2][P.y-2] = 'B';
226:                     currentGameState[P.x-1][P.y-1] = 'p';
227:                     moved = 1;
228:                 }
229:             }
230:             if (arenaLocation.x == P.x-2 && arenaLocation.y == P.y+2) {
231:                 if ((currentGameState[P.x-1][P.y+1] == 'r' ||
232:                     currentGameState[P.x-1][P.y+1] == 'R') &&
233:                     currentGameState[P.x-2][P.y+2] == 'p') {
234:                     currentGameState[P.x-2][P.y+2] = 'B';
235:                     currentGameState[P.x-1][P.y+1] = 'p';
236:                     moved = 1;
237:                 }
238:             }
239:         }
240:     }
241:     if (moved) {
242:         currentGameState[P.x][P.y] = 'p';
243:         finishLine();
244:         deselectPiece();
245:         switchTurn();
246:         stillPlaying = 0;
247:     }
248: }
249:
250: // Interactor
251: void Checkers::deselectPiece(void) {
252:     sf::Vector2i P = getSelectedPawn();
253:     if (P.x == 0 && P.y == 0) return;
254:     if (playerTurn) {
255:         if (currentGameState[P.x][P.y] == 'W')
```

```
256:             currentGameState[P.x][P.y] = 'R';
257:         else
258:             currentGameState[P.x][P.y] = 'r';
259:         stillPlaying = 0;
260:         return;
261:     } else {
262:         if (currentGameState[P.x][P.y] == 'W')
263:             currentGameState[P.x][P.y] = 'B';
264:         else
265:             currentGameState[P.x][P.y] = 'b';
266:         stillPlaying = 0;
267:         return;
268:     }
269: }
270:
271: // Interactor
272: void Checkers::restart(bool& winCondition) {
273:     initializeBase();
274:     playerTurn = 0;
275:     winCondition = 1;
276:     setWinTrue = 0;
277:     stillPlaying = 0;
278: }
279:
280: // Performer
281: void Checkers::playSound(void) {
282:     sf::SoundBuffer buffer;
283:     if (!buffer.loadFromFile("checkers/victory.wav")) exit(1);
284:     sf::Sound sound(buffer);
285:     sound.play();
286:     while (sound.getStatus() == sf::Sound::Playing) {
287:         // Wait for the sound to finish playing
288:     }
289: }
290:
291: // Performer
292: void Checkers::visualMoveAssist(sf::RenderTarget &target) {
293:     sf::Texture star;
294:     if (!star.loadFromFile("checkers/star_icon.png")) exit(1);
295:     sf::Vector2i P = getSelectedPawn();
296:     if (P.x == 0 && P.y == 0) return;
297:     bool fourD = 0, moveable = 0;
298:     if (currentGameState[P.x][P.y] == 'W') fourD = true;
299:     if (playerTurn) {
300:         // red pawn
301:         if (P.x-1 >=0 && P.y-1 >=0) {
302:             if (currentGameState[P.x-1][P.y-1] == 'p') {
303:                 drawStar(target, star, P.x-1, P.y-1);
304:                 moveable = 1;
305:             }
306:         }
307:         if (P.x-2 >= 0 && P.y-2 >= 0) {
308:             if ((currentGameState[P.x-1][P.y-1] == 'b' ||
309:                 currentGameState[P.x-1][P.y-1] == 'B') &&
310:                 currentGameState[P.x-2][P.y-2] == 'p') {
311:                 drawStar(target, star, P.x-2, P.y-2);
312:                 moveable = 1;
313:             }
314:         }
315:         if (P.x-1 >=0 && P.y+1 < 8) {
316:             if (currentGameState[P.x-1][P.y+1] == 'p') {
317:                 drawStar(target, star, P.x-1, P.y+1);
318:                 moveable = 1;
319:             }
320:         }
```

```
321:         if (P.x-2 >= 0 && P.y+2 < 8) {
322:             if ((currentGameState[P.x-1][P.y+1] == 'b' ||
323:                 currentGameState[P.x-1][P.y+1] == 'B') &&
324:                 currentGameState[P.x-2][P.y+2] == 'p') {
325:                 drawStar(target, star, P.x-2, P.y+2);
326:                 moveable = 1;
327:             }
328:         }
329:         if (fourD) {
330:             if (P.x+1 < 8 && P.y-1 >=0) {
331:                 if (currentGameState[P.x+1][P.y-1] == 'p') {
332:                     drawStar(target, star, P.x+1, P.y-1);
333:                     moveable = 1;
334:                 }
335:             }
336:             if (P.x+2 < 8 && P.y-2 >= 0) {
337:                 if ((currentGameState[P.x+1][P.y-1] == 'b' ||
338:                     currentGameState[P.x+1][P.y-1] == 'B') &&
339:                     currentGameState[P.x+2][P.y-2] == 'p') {
340:                     drawStar(target, star, P.x+2, P.y-2);
341:                     moveable = 1;
342:                 }
343:             }
344:             if (P.x+1 < 8 && P.y+1 < 8) {
345:                 if (currentGameState[P.x+1][P.y+1] == 'p') {
346:                     drawStar(target, star, P.x+1, P.y+1);
347:                     moveable = 1;
348:                 }
349:             }
350:             if (P.x+2 < 8 && P.y+2 < 8) {
351:                 if ((currentGameState[P.x+1][P.y+1] == 'b' ||
352:                     currentGameState[P.x+1][P.y+1] == 'B') &&
353:                     currentGameState[P.x+2][P.y+2] == 'p') {
354:                     drawStar(target, star, P.x+2, P.y+2);
355:                     moveable = 1;
356:                 }
357:             }
358:         }
359:     } else {
360:         // black pawn
361:         if (P.x+1 < 8 && P.y-1 >=0) {
362:             if (currentGameState[P.x+1][P.y-1] == 'p') {
363:                 drawStar(target, star, P.x+1, P.y-1);
364:                 moveable = 1;
365:             }
366:         }
367:         if (P.x+2 < 8 && P.y-2 >= 0) {
368:             if ((currentGameState[P.x+1][P.y-1] == 'r' ||
369:                 currentGameState[P.x+1][P.y-1] == 'R') &&
370:                 currentGameState[P.x+2][P.y-2] == 'p') {
371:                 drawStar(target, star, P.x+2, P.y-2);
372:                 moveable = 1;
373:             }
374:         }
375:         if (P.x+1 < 8 && P.y+1 < 8) {
376:             if (currentGameState[P.x+1][P.y+1] == 'p') {
377:                 drawStar(target, star, P.x+1, P.y+1);
378:                 moveable = 1;
379:             }
380:         }
381:         if (P.x+2 < 8 && P.y+2 < 8) {
382:             if ((currentGameState[P.x+1][P.y+1] == 'r' ||
383:                 currentGameState[P.x+1][P.y+1] == 'R') &&
384:                 currentGameState[P.x+2][P.y+2] == 'p') {
385:                 drawStar(target, star, P.x+2, P.y+2);
```

```
  386:                 moveable = 1;
  387:             }
  388:         }
  389:         if (fourD) {
  390:             if (P.x-1 >=0 && P.y-1 >=0) {
  391:                 if (currentGameState[P.x-1][P.y-1] == 'p') {
  392:                     drawStar(target, star, P.x-1, P.y-1);
  393:                     moveable = 1;
  394:                 }
  395:             }
  396:             if (P.x-2 >= 0 && P.y-2 >= 0) {
  397:                 if ((currentGameState[P.x-1][P.y-1] == 'r' ||
  398:                     currentGameState[P.x-1][P.y-1] == 'R') &&
  399:                     currentGameState[P.x-2][P.y-2] == 'p') {
  400:                     drawStar(target, star, P.x-2, P.y-2);
  401:                     moveable = 1;
  402:                 }
  403:             }
  404:             if (P.x-1 >=0 && P.y+1 < 8) {
  405:                 if (currentGameState[P.x-1][P.y+1] == 'p') {
  406:                     drawStar(target, star, P.x-1, P.y+1);
  407:                     moveable = 1;
  408:                 }
  409:             }
  410:             if (P.x-2 >= 0 && P.y+2 < 8) {
  411:                 if ((currentGameState[P.x-1][P.y+1] == 'r' ||
  412:                     currentGameState[P.x-1][P.y+1] == 'R') &&
  413:                     currentGameState[P.x-2][P.y+2] == 'p') {
  414:                     drawStar(target, star, P.x-2, P.y+2);
  415:                     moveable = 1;
  416:                 }
  417:             }
  418:         }
  419:     }
  420:     if (!moveable) {
  421:         bool r = 0, b = 0;
  422:         for (int i = 0; i < BOARD_DIMENSIONS; i++) {
  423:             for (int j = 0; j < BOARD_DIMENSIONS; j++) {
  424:                 if (currentGameState[i][j] == 'r' ||
  425:                     currentGameState[i][j] == 'R')
  426:                     r = 1;
  427:                 if (currentGameState[i][j] == 'b' ||
  428:                     currentGameState[i][j] == 'B')
  429:                     b = 1;
  430:             }
  431:         }
  432:         if (!(r && b)) setWinTrue = 1;
  433:     }
  434: }
  435:
  436: // Performer
  437: void Checkers::drawStar(sf::RenderTarget& target, sf::Texture star, int y
c, int xc) {
  438:     sf::Sprite icon;
  439:     icon.setTexture(star);
  440:     // -8 is an arbitrary offset to center the star on the tile
  441:     icon.setOrigin(-8, -8);
  442:     icon.setPosition(xc*TILE_SIZE+BOARD_OFFSET, yc*TILE_SIZE+BOARD_OFFSET
);
  443:     target.draw(icon);
  444: }
  445:
  446: // Draw game in SFML
  447: void Checkers::draw(sf::RenderTarget& target, sf::RenderStates states) co
nst {
```

```
   448:     // Load piece textures
   449:     sf::Texture blackPiece, blackKing, redPiece, redKing, whitePiece, whi
teKing, woodBacking;
   450:     if (!blackPiece.loadFromFile("checkers/blackpawn.png")) exit(1);
   451:     if (!redPiece.loadFromFile("checkers/redpawn.png")) exit(1);
   452:     if (!whitePiece.loadFromFile("checkers/whitepawn.png")) exit(1);
   453:     if (!blackKing.loadFromFile("checkers/blackking.png")) exit(1);
   454:     if (!redKing.loadFromFile("checkers/redking.png")) exit(1);
   455:     if (!whiteKing.loadFromFile("checkers/whiteking.png")) exit(1);
   456:     if (!woodBacking.loadFromFile("checkers/wood_texture.png")) exit(1);
   457:     sf::Sprite WB;
   458:     WB.setTexture(woodBacking);
   459:     // scale up wood textures
   460:     WB.setScale(1, 1.2);
   461:     // draw backing
   462:     target.draw(WB);
   463:     for (int i = 0; i < BOARD_DIMENSIONS; i++) {
   464:         for (int j = 0; j < BOARD_DIMENSIONS; j++) {
   465:             // Draw states stored in currentGameState
   466:             switch (currentGameState[i][j]) {
   467:                 case '.':
   468:                 {
   469:                     sf::RectangleShape R;
   470:                     R.setSize(Vector2f(TILE_SIZE, TILE_SIZE));
   471:                     R.setFillColor(sf::Color::Red);
   472:                     R.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE_
SIZE + BOARD_OFFSET);
   473:                     target.draw(R);
   474:                     break;
   475:                 }
   476:                 case 'p':
   477:                 {
   478:                     drawBackingRectangle(target, j, i);
   479:                     break;
   480:                 }
   481:                 case 'r':
   482:                 {
   483:                     drawBackingRectangle(target, j, i);
   484:                     sf::Sprite RP;
   485:                     RP.setTexture(redPiece);
   486:                     RP.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE
_SIZE + BOARD_OFFSET);
   487:                     target.draw(RP);
   488:                     break;
   489:                 }
   490:                 case 'b':
   491:                 {
   492:                     drawBackingRectangle(target, j, i);
   493:                     sf::Sprite BP;
   494:                     BP.setTexture(blackPiece);
   495:                     BP.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE
_SIZE + BOARD_OFFSET);
   496:                     target.draw(BP);
   497:                     break;
   498:                 }
   499:                 case 'w':
   500:                 {
   501:                     drawBackingRectangle(target, j, i);
   502:                     sf::Sprite WP;
   503:                     WP.setTexture(whitePiece);
   504:                     // scale down white pieces to 64 x 64 pixels
   505:                     WP.setScale(0.6, 0.6);
   506:                     WP.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE
_SIZE + BOARD_OFFSET);
   507:                     target.draw(WP);
```

```
  508:                     break;
  509:                 }
  510:                 case 'R':
  511:                 {
  512:                     drawBackingRectangle(target, j, i);
  513:                     sf::Sprite RP;
  514:                     RP.setTexture(redKing);
  515:                     RP.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE
_SIZE + BOARD_OFFSET);
  516:                     target.draw(RP);
  517:                     break;
  518:                 }
  519:                 case 'B':
  520:                 {
  521:                     drawBackingRectangle(target, j, i);
  522:                     sf::Sprite BP;
  523:                     BP.setTexture(blackKing);
  524:                     BP.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE
_SIZE + BOARD_OFFSET);
  525:                     target.draw(BP);
  526:                     break;
  527:                 }
  528:                 case 'W':
  529:                 {
  530:                     drawBackingRectangle(target, j, i);
  531:                     sf::Sprite WP;
  532:                     WP.setTexture(whiteKing);
  533:                     // scale down white pieces to 64 x 64 pixels
  534:                     WP.setScale(0.6, 0.6);
  535:                     WP.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE
_SIZE + BOARD_OFFSET);
  536:                     target.draw(WP);
  537:                     break;
  538:                 }
  539:             }
  540:         }
  541:     }
  542:     sf::Font font;
  543:     font.loadFromFile("checkers/arial.ttf");
  544:     sf::Text turnText("", font, 40);
  545:     turnText.setFillColor(sf::Color::White);
  546:     turnText.setPosition(Vector2f(10, 0));
  547:     if (playerTurn) {
  548:         // Red turn
  549:         turnText.setString("R");
  550:     } else {
  551:         // Black turn
  552:         turnText.setString("B");
  553:     }
  554:     target.draw(turnText);
  555: }
  556:
  557: // Initialize game storage vectors
  558: void Checkers::initializeBase(void) {
  559:     currentGameState.resize(BOARD_DIMENSIONS);
  560:     for (int i = 0; i < BOARD_DIMENSIONS; i++) {
  561:         currentGameState[i].resize(BOARD_DIMENSIONS);
  562:         for (int j = 0; j < BOARD_DIMENSIONS; j++) {
  563:             // Draw red background tiles
  564:             if (((j + (i%2))%2)) {
  565:                 if (i <= 2) {
  566:                     // Black piece represented as 'b' (black)
  567:                     // placed on lines 0 - 2
  568:                     currentGameState[i][j] = 'b';
  569:                 } else if (i >= 5) {
```

```
  570:                     // Red piece represented as 'r' (red)
  571:                     // placed on lines 5 - 7
  572:                     currentGameState[i][j] = 'r';
  573:                 } else {
  574:                     // Black background tile represented 'p' (playable)
  575:                     currentGameState[i][j] = 'p';
  576:                 }
  577:             } else {
  578:                 // Red background tile represented as '.' (invalid)
  579:                 currentGameState[i][j] = '.';
  580:             }
  581:         }
  582:     }
  583: }
  584:
  585: // Automatically king pawns at respective finish lines
  586: void Checkers::finishLine(void) {
  587:     for (int i = 0; i < BOARD_DIMENSIONS; i++) {
  588:         if (!(i%2)) {
  589:             // black pieces - 7 is last row
  590:             if (currentGameState[7][i] == 'b') {
  591:                 currentGameState[7][i] = 'B';
  592:                 return;
  593:             }
  594:         } else {
  595:             // red pieces - 0 is first row
  596:             if (currentGameState[0][i] == 'r') {
  597:                 currentGameState[0][i] = 'R';
  598:                 return;
  599:             }
  600:         }
  601:     }
  602: }
  603:
  604: // Helper function
  605: bool mouseInGameBounds(sf::Vector2i mouseLocation) {
  606:     if ((mouseLocation.x >= BOARD_OFFSET && mouseLocation.x <=
  607:         TILE_SIZE * BOARD_DIMENSIONS + BOARD_OFFSET) &&
  608:         (mouseLocation.y >= BOARD_OFFSET && mouseLocation.y <=
  609:         TILE_SIZE * BOARD_DIMENSIONS + BOARD_OFFSET)) return 1;
  610:     return 0;
  611: }
  612:
  613: // Performing helper function
  614: void drawBackingRectangle(sf::RenderTarget& target, int x, int y) {
  615:     sf::RectangleShape B;
  616:     B.setSize(Vector2f(TILE_SIZE, TILE_SIZE));
  617:     B.setFillColor(sf::Color::Black);
  618:     B.setPosition(x * TILE_SIZE + BOARD_OFFSET, y * TILE_SIZE + BOARD_OFF
SET);
  619:     target.draw(B);
  620: }
```