```
    1: // Copyright 2023 Thomas O'Connor
    2: #include "Sokoban.hpp"
    3:
    4: // Extract the entire gameState into the Sokoban object
    5: Sokoban& operator>>(Sokoban& game, std::ifstream& file) {
    6:     file >> game._h; file >> game._w;
    7:     game.face.resize(game.turn + 1);
    8:     game.face[0] = DOWN;
    9:     game.allGameStates.resize(game.turn + 1);
   10:     // Given game dimensions, resize 2D vector and input from file
   11:     game.allGameStates[game.turn].resize(game._h);
   12:     for (int i = 0; i < game._h; i++) {
   13:         game.allGameStates[game.turn][i].resize(game._w);
   14:         for (int j = 0; j < game._w; j++) {
   15:             file >> game.allGameStates[game.turn][i][j];
   16:         }
   17:     }
   18:     return game;
   19: }
   20:
   21: bool Sokoban::isWon() const {
   22:     auto it = std::find_if(allGameStates[turn].begin(), allGameStates[tur
n].end(),
   23:         [](const std::vector<char>& row) { return std::any_of(row.begin()
, row.end(),
   24:         [](char c) { return c == 'A' || c == '2'; });
   25:     });
   26:     if (it != allGameStates[turn].end()) return false;
   27:     else
   28:     return true;
   29: }
   30:
   31: // Moves the player given a direction
   32: void Sokoban::movePlayer(Direction dir) {
   33:     for (int i = 0; i < _h; i++) {
   34:         for (int j = 0; j < _w; j++) {
   35:             // When find player, check for no obstructions and execute ac
tion
   36:             if (allGameStates[turn][i][j] == '@' || allGameStates[turn][i
][j] == '2') {
   37:                 if (noObstructions(dir, j, i)) {
   38:                     // If no obstructions, create a new turn and a new ga
me state
   39:                     turn++;
   40:                     allGameStates.resize(turn + 1);
   41:                     allGameStates[turn] = deepCopy(allGameStates[turn - 1
]);
   42:                     face.resize(turn + 1);
   43:                     face[turn] = dir;
   44:                     if (allGameStates[turn][i][j] == '2') allGameStates[t
urn][i][j] = 'a';
   45:                     else
   46:                     allGameStates[turn][i][j] = '.';
   47:                     if (canPushBox(dir, j, i)) pushBox(dir, j, i);
   48:                     switch (dir) {
   49:                         // UP
   50:                         case 0:
   51:                          if (allGameStates[turn][i-1][j] == 'a') allGameS
tates[turn][i-1][j] = '2';
   52:                          else
   53:                          allGameStates[turn][i-1][j] = '@';
   54:                          break;
   55:                         // LEFT
   56:                         case 1:
   57:                          if (allGameStates[turn][i][j-1] == 'a') allGameS
```

```
tates[turn][i][j-1] = '2';
   58:                          else
   59:                          allGameStates[turn][i][j-1] = '@';
   60:                          break;
   61:                         // DOWN
   62:                         case 2:
   63:                          if (allGameStates[turn][i+1][j] == 'a') allGameS
tates[turn][i+1][j] = '2';
   64:                          else
   65:                          allGameStates[turn][i+1][j] = '@';
   66:                          break;
   67:                         // RIGHT
   68:                         case 3:
   69:                          if (allGameStates[turn][i][j+1] == 'a') allGameS
tates[turn][i][j+1] = '2';
   70:                          else
   71:                          allGameStates[turn][i][j+1] = '@';
   72:                          break;
   73:                     }
   74:                     return;
   75:                 }
   76:             }
   77:         }
   78:     }
   79: }
   80:
   81: // Restarts the game when 'R' is pressed
   82: void Sokoban::restart(sf::Clock& clock) {
   83:     turn = 0;
   84:     clock.restart();
   85:     }
   86:
   87: // Reverts back to the previous game state
   88: void Sokoban::undo() { if (turn) turn--; }
   89:
   90: void Sokoban::playSound() {
   91:     for (int i = 0; i < _h; i++) {
   92:         for (int j = 0; j < _w; j++) {
   93:             if (allGameStates[turn][i][j] == 'A' || allGameStates[turn][i
][j] == '2') return;
   94:         }
   95:     }
   96:     sf::SoundBuffer buffer;
   97:     if (!buffer.loadFromFile("sokoban/victory.wav")) exit(1);
   98:     sf::Sound sound(buffer);
   99:     sound.play();
  100:     while (sound.getStatus() == sf::Sound::Playing) {
  101:         // Wait for the sound to finish playing
  102:     }
  103: }
  104:
  105: // Check if there is a wall or double boxes
  106: bool Sokoban::noObstructions(Direction dir, int w, int h) const {
  107:     switch (dir) {
  108:         // UP
  109:         case 0:
  110:          // Out of game bounds
  111:          if (!h) return false;
  112:          // Wall
  113:          if (allGameStates[turn][h-1][w] == '#') return false;
  114:          // Full Storage
  115:          if (allGameStates[turn][h-1][w] == '1') return false;
  116:          // Double boxes / Box - wall / Box - storage / Box - EOL
  117:          if (allGameStates[turn][h-1][w] == 'A' && (h == 1)) return false
;
```

```
  118:          if (allGameStates[turn][h-1][w] == 'A' && (
  119:              allGameStates[turn][h-2][w] == 'A' ||
  120:              allGameStates[turn][h-2][w] == '#' ||
  121:              allGameStates[turn][h-2][w] == '1')) return false;
  122:          break;
  123:         // LEFT
  124:         case 1:
  125:          // Out of game bounds
  126:          if (!w) return false;
  127:          // Full Storage
  128:          if (allGameStates[turn][h][w-1] == '1') return false;
  129:          // Check for wall
  130:          if (allGameStates[turn][h][w-1] == '#') return false;
  131:          // Double boxes / Box - wall / Box - storage / Box - EOL
  132:          if (allGameStates[turn][h][w-1] == 'A' && (
  133:              allGameStates[turn][h][w-2] == 'A' ||
  134:              allGameStates[turn][h][w-2] == '#' ||
  135:              allGameStates[turn][h][w-2] == '1' ||
  136:              (w == 1))) return false;
  137:          break;
  138:         // DOWN
  139:         case 2:
  140:          // Out of game bounds
  141:          if (!(_h-h-1)) return false;
  142:          // Full Storage
  143:          if (allGameStates[turn][h+1][w] == '1') return false;
  144:          // Wall
  145:          if (allGameStates[turn][h+1][w] == '#') return false;
  146:          // Double boxes / Box - wall / Box - storage / Box - EOL
  147:          if (allGameStates[turn][h+1][w] == 'A' && (h+2 == _h)) return fa
lse;
  148:          if (allGameStates[turn][h+1][w] == 'A' && (
  149:              allGameStates[turn][h+2][w] == 'A' ||
  150:              allGameStates[turn][h+2][w] == '#' ||
  151:              allGameStates[turn][h+2][w] == '1')) return false;
  152:          break;
  153:         // RIGHT
  154:         case 3:
  155:          // Out of game bounds
  156:          if (!(_w-w-1)) return false;
  157:          // Full Storage
  158:          if (allGameStates[turn][h][w+1] == '1') return false;
  159:          // Wall
  160:          if (allGameStates[turn][h][w+1] == '#') return false;
  161:          // Double boxes / Box - wall / Box - storage / Box - EOL
  162:          if (allGameStates[turn][h][w+1] == 'A' && (
  163:              allGameStates[turn][h][w+2] == 'A' ||
  164:              allGameStates[turn][h][w+2] == '#' ||
  165:              allGameStates[turn][h][w+2] == '1' ||
  166:              (w+2 == _w))) return false;
  167:          break;
  168:     }
  169:     return true;
  170: }
  171:
  172: bool Sokoban::canPushBox(Direction dir, int w, int h) const {
  173:     switch (dir) {
  174:         // UP
  175:         case 0:
  176:          // Box - empty / Box - storage
  177:          if (allGameStates[turn][h-1][w] == 'A' && (
  178:              allGameStates[turn][h-2][w] == '.' ||
  179:              allGameStates[turn][h-2][w] == 'a')) return true;
  180:          break;
  181:         // LEFT
```

```
  182:         case 1:
  183:          // Box - empty / Box - storage
  184:          if (allGameStates[turn][h][w-1] == 'A' && (
  185:              allGameStates[turn][h][w-2] == '.' ||
  186:              allGameStates[turn][h][w-2] == 'a')) return true;
  187:          break;
  188:         // DOWN
  189:         case 2:
  190:          // Box - empty / Box - storage
  191:          if (allGameStates[turn][h+1][w] == 'A' && (
  192:              allGameStates[turn][h+2][w] == '.' ||
  193:              allGameStates[turn][h+2][w] == 'a')) return true;
  194:          break;
  195:         // RIGHT
  196:         case 3:
  197:          // Box - empty / Box - storage
  198:          if (allGameStates[turn][h][w+1] == 'A' && (
  199:              allGameStates[turn][h][w+2] == '.' ||
  200:              allGameStates[turn][h][w+2] == 'a')) return true;
  201:          break;
  202:     }
  203:     return false;
  204: }
  205:
  206: void Sokoban::pushBox(Direction dir, int w, int h) {
  207:     switch (dir) {
  208:         // UP
  209:         case 0:
  210:          if (allGameStates[turn][h-2][w] == '.') allGameStates[turn][h-2]
[w] = 'A';
  211:          if (allGameStates[turn][h-2][w] == 'a') allGameStates[turn][h-2]
[w] = '1';
  212:          break;
  213:         // LEFT
  214:         case 1:
  215:          if (allGameStates[turn][h][w-2] == '.') allGameStates[turn][h][w
-2] = 'A';
  216:          if (allGameStates[turn][h][w-2] == 'a') allGameStates[turn][h][w
-2] = '1';
  217:          break;
  218:         // DOWN
  219:         case 2:
  220:          if (allGameStates[turn][h+2][w] == '.') allGameStates[turn][h+2]
[w] = 'A';
  221:          if (allGameStates[turn][h+2][w] == 'a') allGameStates[turn][h+2]
[w] = '1';
  222:          break;
  223:         // RIGHT
  224:         case 3:
  225:          if (allGameStates[turn][h][w+2] == '.') allGameStates[turn][h][w
+2] = 'A';
  226:          if (allGameStates[turn][h][w+2] == 'a') allGameStates[turn][h][w
+2] = '1';
  227:          break;
  228:         return;
  229:     }
  230: }
  231:
  232: // Overload the virtual draw function:
  233: // Load required textures and display in window at prescribed locations
  234: void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) con
st {
  235:     sf::Texture Wall, Box, WinBox, Empty, Storage, ManDown, ManUp, ManLef
t, ManRight;
  236:     if (!Wall.loadFromFile("sokoban/block_06.png")) exit(1);
```

```
237:     if (!Box.loadFromFile("sokoban/crate_03.png")) exit(1);
238:     if (!WinBox.loadFromFile("sokoban/crate_04.png")) exit(1);
239:     if (!Empty.loadFromFile("sokoban/ground_01.png")) exit(1);
240:     if (!Storage.loadFromFile("sokoban/environment_03.png")) exit(1);
241:     if (!ManDown.loadFromFile("sokoban/player_05.png")) exit(1);
242:     if (!ManUp.loadFromFile("sokoban/player_08.png")) exit(1);
243:     if (!ManLeft.loadFromFile("sokoban/player_20.png")) exit(1);
244:     if (!ManRight.loadFromFile("sokoban/player_17.png")) exit(1);
245:
246:     for (int i = 0; i < _h; i++) {
247:         for (int j = 0; j < _w; j++) {
248:             sf::RectangleShape tile;
249:             tile.setSize(sf::Vector2f(TILE_SIZE, TILE_SIZE));
250:             tile.setPosition(j * TILE_SIZE, i * TILE_SIZE);
251:             // Draw wall tile
252:             if (allGameStates[turn][i][j] == '#') {
253:                 tile.setTexture(&Wall);
254:             // Draw empty tile
255:             } else if (allGameStates[turn][i][j] == '.') {
256:                 tile.setTexture(&Empty);
257:             // Draw background tile and overlay
258:             } else {
259:                 sf::RectangleShape backTile;
260:                 backTile.setSize(sf::Vector2f(TILE_SIZE, TILE_SIZE));
261:                 backTile.setPosition(j * TILE_SIZE, i * TILE_SIZE);
262:                 backTile.setTexture(&Empty);
263:                 target.draw(backTile);
264:                 switch (allGameStates[turn][i][j]) {
265:                     case 'a':
266:                      tile.setTexture(&Storage);
267:                      break;
268:                     case 'A':
269:                      tile.setTexture(&Box);
270:                      break;
271:                     case '1':
272:                      tile.setTexture(&WinBox);
273:                      break;
274:                     default:
275:                      // Based on face turn, select appropriate texture
276:                      if (face[turn] == DOWN) tile.setTexture(&ManDown);
277:                      if (face[turn] == UP) tile.setTexture(&ManUp);
278:                      if (face[turn] == LEFT) tile.setTexture(&ManLeft);
279:                      if (face[turn] == RIGHT) tile.setTexture(&ManRight);
280:                      break;
281:                 }
282:             }
283:             target.draw(tile);
284:         }
285:     }
286: }
287:
288: // Function that displays time in upper-left corner
289: void Sokoban::drawElapsingTime(sf::RenderWindow &window, sf::Clock &clock
) {
290:     sf::Time elapsed = clock.getElapsedTime();
291:     int minutes = elapsed.asSeconds() / 60;
292:     int seconds = static_cast<int>(elapsed.asSeconds()) % 60;
293:
294:     std::string timeString = std::to_string(minutes) + ":" +
295:     (seconds < 10 ? "0" : "") + std::to_string(seconds);
296:
297:     sf::Font font;
298:     font.loadFromFile("sokoban/arial.ttf");
299:     sf::Text timeText(timeString, font, 30);
300:     timeText.setFillColor(sf::Color::White);
```

```
301:
302:     window.draw(timeText);
303: }
304:
305: // non-member helper functions:
306: // performs a deep copy of the game state
307: vector<vector<char>> deepCopy(const vector<vector<char>> &original) {
308:     vector<vector<char>> copy;
309:     for (const auto &row : original) copy.push_back(row);
310:     return copy;
311: }
```