PS4: Sokoban

Assignment Overview

This project involved developing a fully functional version of the classic tile-based puzzle game Sokoban, divided into two parts. In Part A, the focus was on building a Sokoban class to read level files, represent the game board, and render the correct textures (walls, floors, boxes, storage, and the player) using SFML. The level was drawn dynamically based on the provided file, and key elements like the player's location and the size of the board were correctly interpreted. In Part B, gameplay logic was implemented. This included support for moving the player with the keyboard, pushing boxes, detecting collisions with walls and other boxes, and determining when the player has won. Resetting the level and displaying a victory message were also added, and extra credit was completed by including directional player sprites, a move counter, and an undo feature.

Key Algorithms / Data Structures / OO Designs

The program is organized within a SB namespace, with a single class Sokoban that encapsulates all state and logic for a game session. Internally, the level data is stored as a std::vector<std::vector<char>> grid. The draw() method is overridden from sf::Drawable, and various SFML textures and sprites are initialized once and reused efficiently to render each tile. Movement logic is handled in movePlayer(Direction dir) which computes directional deltas, checks for collisions, and supports pushing boxes only when valid. Game state is tracked using a custom GameState struct and a std::stack to implement undo functionality. A win condition is determined by checking if all storage locations are filled with boxes. Lambda expressions and STL algorithms like std::all_of are used to check game state concisely. Operator overloading for >> and << enables easy reading and saving of game boards, and Boost unit tests were implemented to check all edge cases like pushing into walls, winning, or moving off-screen.

What I Learned

Through PS4, I learned how to use SFML to load and render 2D tile maps using textures and sprites, and how to parse level data from structured input files and convert it into an internal grid representation. I implemented collision detection, conditional movement, and game rules for interactive gameplay, and created modular, object-oriented code that cleanly separates drawing, logic, and input handling. I applied undo functionality using a stack of previous game states, and used lambda expressions and STL algorithms such as std::all_of for compact game condition checks. I also built unit tests using the Boost testing library to validate core game mechanics and handled dynamic window sizing based on game dimensions to maintain readable, scalable code.

What Doesn't Work

While the game is fully playable and implements most required features, several edge cases in the autograder reveal limitations in the win condition logic. In all cases, isWon() returns true prematurely, possibly due to lenient win condition logic not fully verifying both box and storage alignment. Unfortunately I was not able to debug this portion, but the game thankfully works as intended otherwise.

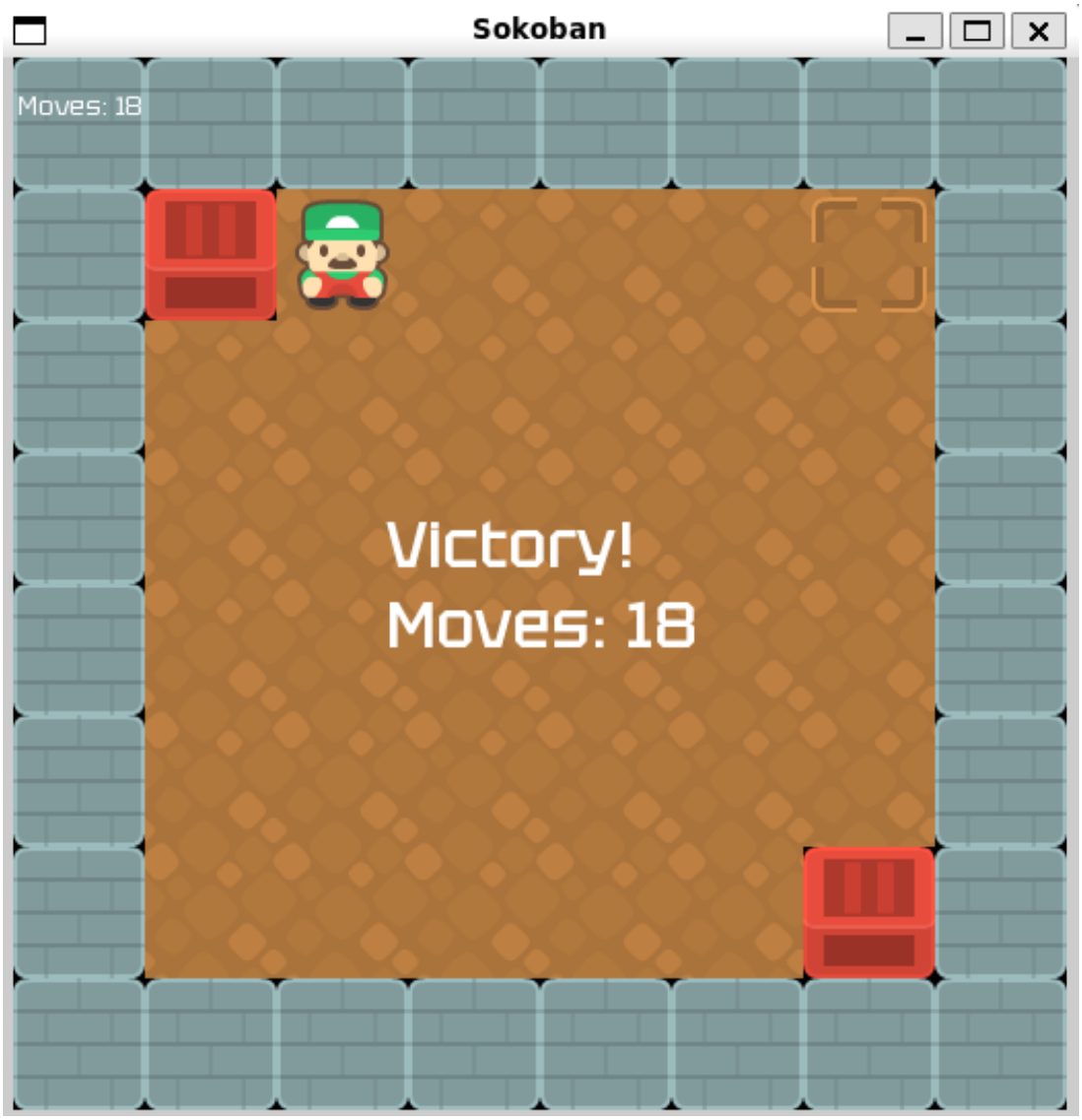

Figure 5: Example Output

Source Code

```
Makefile:
# Compiler and flags
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework

# Dependencies and objects
DEPS = Sokoban.hpp
OBJECTS = Sokoban.o

# Program name
PROGRAM = Sokoban
```

```
TEST = test

.PHONY: all clean lint

# Default target: build the program and static library
all: $(PROGRAM) $(TEST) Sokoban.a

# Rule to build the program
$(PROGRAM): main.o Sokoban.a
	$(CC) $(CFLAGS) -o $@ main.o Sokoban.a $(LIB)

$(TEST): test.o $(OBJECTS)
	$(CC) $(CFLAGS) -o $@ $^ $(LIB)
# Rule to create the static library
Sokoban.a: $(OBJECTS)
	ar rcs $@ $^

# Rule to compile object files
%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $< -o $@

# Clean up build files
clean:
	rm -f *.o $(PROGRAM) $(TEST) Sokoban.a

# Run cpplint for static analysis
lint:
	cpplint *.cpp *.hpp

main.cpp:
// COPYRIGHT 2025 Kaden Gardiner
#include "Sokoban.hpp"

int main(int argc, char* argv[]) {
    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <level_file>\n";
        return 1;
    }

    std::string filePath = "sokoban/";
    filePath += argv[1];
    std::ifstream levelFile(filePath);
    if (!levelFile) {
        std::cerr << "Error: Could not open file " << filePath << "\n";
        return 1;
    }

    SB::Sokoban game(0, 0, {});
    levelFile >> game;
    sf::RenderWindow window(sf::VideoMode(game.width() * SB::Sokoban::
       TILE_SIZE,
    game.height() * SB::Sokoban::TILE_SIZE), "Sokoban");
```

```
64
65      while (window.isOpen()) {
66          sf::Event event;
67          while (window.pollEvent(event)) {
68              if (event.type == sf::Event::Closed) {
69                  window.close();
70              }
71              if (event.type == sf::Event::KeyPressed) {
72                  if (!game.isWon()) {
73                      if (event.key.code == sf::Keyboard::W) {
74                          game.movePlayer(SB::Direction::Up);
75                      } else if (event.key.code == sf::Keyboard::S) {
76                          game.movePlayer(SB::Direction::Down);
77                      } else if (event.key.code == sf::Keyboard::A) {
78                          game.movePlayer(SB::Direction::Left);
79                      } else if (event.key.code == sf::Keyboard::D) {
80                          game.movePlayer(SB::Direction::Right);
81                      }
82                  }
83                  if (event.key.code == sf::Keyboard::R) {
84                      game.reset();
85                  } else if (event.key.code == sf::Keyboard::U) {
86                      game.undo();
87                  }
88              }
89          }
90
91          if (game.isWon() && !game.isGameWon()) {
92              std::cout << "Victory was achieved in "
93              << game.moveCount() << " moves" << std::endl;
94              game.setGameWon(true);
95          }
96
97          window.clear();
98          window.draw(game);
99          window.display();
100     }
101
102     return 0;
103 }
104
105 Sokoban.hpp:
106 // COPYRIGHT 2025 Kaden Gardiner
107 #pragma once
108
109 #include <iostream>
110 #include <vector>
111 #include <fstream>
112 #include <string>
113 #include <stack>
114 #include <algorithm>
115 #include <SFML/Graphics.hpp>
```

```
116 #include <SFML/Audio.hpp>
117
118 namespace SB {
119 enum class Direction {
120     Up, Down, Left, Right
121 };
122
123 struct GameState {
124     sf::Vector2u playerPosition;
125     std::vector<std::vector<char>> board;
126     size_t moveCount;
127 };
128
129 class Sokoban : public sf::Drawable {
130  public:
131     static const int TILE_SIZE = 64;
132     Sokoban() : _height(0), _width(0), _board(),
133    _initialBoard(), _moveCount(0) {}
134     Sokoban(size_t height, size_t width,
135       const std::vector<std::vector<char>>& board);
136     // Sokoban(const std::string&);  // Optional
137     // unsigned int pixelHeight() const;  // Optional
138     // unsigned int pixelWidth() const;  // Optional
139
140     unsigned int height() const {return _height;}
141     unsigned int width() const {return _width;}
142     sf::Vector2u playerLoc() const;
143     std::vector<sf::Vector2u> storageLoc() const;
144     std::vector<sf::Vector2u> boxLoc() const;
145     bool isWon() const;
146     void movePlayer(Direction dir);
147     void reset();
148     size_t moveCount() const {return _moveCount;}
149     bool isGameWon() const {return _isGameWon;}
150     void setGameWon(bool isGameWon) {_isGameWon = isGameWon;}
151     void setPlayerTexture(Direction dir);
152     void undo();  // Optional XC
153     // void redo(); // Optional XC
154     friend std::ostream& operator<<(std::ostream& out, const Sokoban& s);
155     friend std::istream& operator>>(std::istream& in, Sokoban& s);
156
157  protected:
158     void draw(sf::RenderTarget& target, sf::RenderStates states) const
        override;
159
160  private:
161     size_t _height;
162     size_t _width;
163     std::stack<GameState> _undoStack;
164     std::vector<std::vector<char>> _board;
165     std::vector<std::vector<char>> _initialBoard;
166     sf::Vector2u _playerPosition;
```

```
    size_t _moveCount;
    bool _isGameWon;
    sf::Texture _wall;
    sf::Texture _ground;
    sf::Texture _playerDown;
    sf::Texture _playerUp;
    sf::Texture _playerLeft;
    sf::Texture _playerRight;
    sf::Texture _box;
    sf::Texture _storage;
    sf::Font _font;
    mutable sf::Text _moveText;
    mutable sf::Sprite _player;
};
}  // namespace SB

Sokoban.cpp:
// COPYRIGHT 2025 Kaden Gardiner
#include "Sokoban.hpp"

SB::Sokoban::Sokoban(size_t height, size_t width,
    const std::vector<std::vector<char>>& board)
: _height(height), _width(width), _board(board),
_initialBoard(board),  _moveCount(0), _isGameWon(false) {
    if (!_ground.loadFromFile("sokoban/ground_01.png")) {
        std::cerr << "Error: Could not load ground_01.png\n";
    }
    if (!_wall.loadFromFile("sokoban/block_06.png")) {
        std::cerr << "Error: Could not load wall_06.png\n";
    }
    if (!_playerDown.loadFromFile("sokoban/player_05.png")) {
        std::cerr << "Error: Could not load player_05.png\n";
    }
    if (!_playerUp.loadFromFile("sokoban/player_08.png")) {
        std::cerr << "Error: Could not load player_05.png\n";
    }
    if (!_playerLeft.loadFromFile("sokoban/player_20.png")) {
        std::cerr << "Error: Could not load player_05.png\n";
    }
    if (!_playerRight.loadFromFile("sokoban/player_17.png")) {
        std::cerr << "Error: Could not load player_05.png\n";
    }
    if (!_box.loadFromFile("sokoban/crate_03.png")) {
        std::cerr << "Error: Could not load crate_03.png\n";
    }
    if (!_storage.loadFromFile("sokoban/ground_04.png")) {
        std::cerr << "Error: Could not load ground_04.png\n";
    }
    if (!_font.loadFromFile("font.ttf")) {
        std::cerr << "Error: Could not load arial.ttf\n";
    }
    _moveText.setFont(_font);
```

```
219     _moveText.setCharacterSize(13);
220     _moveText.setFillColor(sf::Color::White);
221     _moveText.setPosition(2, 15);
222     _player.setTexture(_playerDown);
223 }
224
225 void SB::Sokoban::draw(sf::RenderTarget& target,
226 sf::RenderStates states) const {
227     sf::Sprite groundSprite(_ground);
228     sf::Sprite wallSprite(_wall);
229     sf::Sprite boxSprite(_box);
230     sf::Sprite storageSprite(_storage);
231
232     for (size_t row = 0; row < _height; row ++) {
233         for (size_t col = 0; col < _width; col++) {
234             char cell = _board[row][col];
235             if (cell != '#' && cell != 'A' && cell != 'a' && cell != '1')
                    {
236                 groundSprite.setPosition(col * TILE_SIZE, row * TILE_SIZE)
                        ;
237                 target.draw(groundSprite);
238             }
239             if (cell == '#') {
240                 wallSprite.setPosition(col * TILE_SIZE, row * TILE_SIZE);
241                 target.draw(wallSprite);
242             }
243         }
244     }
245     for (const auto& boxPosition : boxLoc()) {
246         boxSprite.setPosition(boxPosition.x * TILE_SIZE,
247             boxPosition.y * TILE_SIZE);
248         target.draw(boxSprite);
249     }
250     for (const auto& storagePosition : storageLoc()) {
251         storageSprite.setPosition(storagePosition.x * TILE_SIZE,
252             storagePosition.y * TILE_SIZE);
253         target.draw(storageSprite);
254     }
255     try {
256         sf::Vector2u playerPosition = playerLoc();
257         _player.setPosition(playerPosition.x * TILE_SIZE,
258             playerPosition.y * TILE_SIZE);
259         target.draw(_player);
260     } catch (const std::exception& e) {
261         std::cerr << e.what() << std::endl;
262     }
263     _moveText.setString("Moves: " + std::to_string(_moveCount));
264     target.draw(_moveText);
265     if (_isGameWon) {
266         sf::Text victoryText;
267         victoryText.setFont(_font);
268         victoryText.setCharacterSize(25);
```

```
269         victoryText.setFillColor(sf::Color::White);
270         victoryText.setStyle(sf::Text::Bold);
271         victoryText.setString("           Victory!\n         Moves: "
272             + std::to_string(_moveCount) + "\n    Press 'R' to reset");
273         sf::FloatRect textBounds = victoryText.getLocalBounds();
274         victoryText.setOrigin(textBounds.left + textBounds.width / 2.0f,
275                               textBounds.top + textBounds.height / 2.0f);
276         sf::Vector2u windowSize = target.getSize();
277         victoryText.setPosition(windowSize.x / 2.0f, windowSize.y / 2.0f);
278         target.draw(victoryText);
279     }
280 }
281
282 void SB::Sokoban::setPlayerTexture(Direction dir) {
283     switch (dir) {
284         case Direction::Up:
285             _player.setTexture(_playerUp);
286             break;
287         case Direction::Down:
288             _player.setTexture(_playerDown);
289             break;
290         case Direction::Left:
291             _player.setTexture(_playerLeft);
292             break;
293         case Direction::Right:
294             _player.setTexture(_playerRight);
295             break;
296     }
297 }
298
299 void SB::Sokoban::movePlayer(Direction dir) {
300     _undoStack.push({playerLoc(), _board, _moveCount});
301
302     sf::Vector2u playerPos = playerLoc();
303     int x = playerPos.x;
304     int y = playerPos.y;
305     int dx = 0, dy = 0;
306
307     switch (dir) {
308         case Direction::Up:    dy = -1; break;
309         case Direction::Down:  dy = 1; break;
310         case Direction::Left:  dx = -1; break;
311         case Direction::Right: dx = 1; break;
312     }
313     int newX = x + dx;
314     int newY = y + dy;
315
316     auto isInBounds = [this](int x, int y) {
317         return x >= 0 && x < static_cast<int>(_width) &&
318         y >= 0 && y < static_cast<int>(_height);
319     };
320
```

```
321     if (!isInBounds(newX, newY)) {
322         return;
323     }
324     char targetCell = _board[newY][newX];
325     if (targetCell == '#') {
326         return;
327     }
328     if (targetCell == 'A' || targetCell == '1') {
329         int newBoxX = newX + dx;
330         int newBoxY = newY + dy;
331         if (!isInBounds(newBoxX, newBoxY)) {
332             return;
333         }
334         char boxTargetCell = _board[newBoxY][newBoxX];
335         if (boxTargetCell == '.' || boxTargetCell == 'a') {
336             _board[newBoxY][newBoxX] = 'A';
337             _board[newY][newX] = (_board[newY][newX] == 'A') ? 'a' : '.';
338         } else {
339             return;
340         }
341     }
342     char currentCell = _board[y][x];
343     _board[y][x] = (currentCell == '+') ? 'a' : '.';
344     _board[newY][newX] = (targetCell == 'a') ? '+' : '@';
345     _playerPosition = {static_cast<unsigned int>(newX),
346         static_cast<unsigned int>(newY)};
347     ++_moveCount;
348 }
349
350 void SB::Sokoban::reset() {
351     _board = _initialBoard;
352     _moveCount = 0;
353     _isGameWon = false;
354     _player.setTexture(_playerDown);
355     _playerPosition = playerLoc();
356 }
357
358 bool SB::Sokoban::isWon() const {
359     auto storages = storageLoc();
360     auto boxes = boxLoc();
361
362     bool allStorageOccupied = std::all_of(
363         storages.begin(),
364         storages.end(),
365         [this](const sf::Vector2u& storage) {
366             return _board[storage.y][storage.x] == '1';
367         });
368
369     bool allBoxesOnStorage = std::all_of(
370         boxes.begin(),
371         boxes.end(),
372         [this](const sf::Vector2u& box) {
```

```
373             return _initialBoard[box.y][box.x] == 'a';
374         });
375
376     return allStorageOccupied || allBoxesOnStorage;
377 }
378
379 void SB::Sokoban::undo() {
380     if (_undoStack.empty()) {
381         return;
382     }
383     GameState lastState = _undoStack.top();
384     _undoStack.pop();
385     _board = lastState.board;
386     _moveCount = lastState.moveCount;
387     _playerPosition = lastState.playerPosition;
388     _player.setPosition(_playerPosition.x * TILE_SIZE,
389         _playerPosition.y * TILE_SIZE);
390     _isGameWon = false;
391 }
392
393 sf::Vector2u SB::Sokoban::playerLoc() const {
394     for (size_t row = 0; row < _height; row++) {
395         for (size_t col = 0; col < _width; col++) {
396             if (_board[row][col] == '@' || _board[row][col] == '+') {
397                 return {static_cast<unsigned int>(col),
398                     static_cast<unsigned int>(row)};
399             }
400         }
401     }
402     throw std::runtime_error("Player not found on the board");
403 }
404
405 std::vector<sf::Vector2u> SB::Sokoban::boxLoc() const {
406     std::vector<sf::Vector2u> boxPositions;
407     for (size_t row = 0; row < _height; ++row) {
408         for (size_t col = 0; col < _width; ++col) {
409             if (_board[row][col] == 'A' || _board[row][col] == '1') {
410                 boxPositions.push_back({static_cast<unsigned int>(col),
411                     static_cast<unsigned int>(row)});
412             }
413         }
414     }
415     return boxPositions;
416 }
417
418 std::vector<sf::Vector2u> SB::Sokoban::storageLoc() const {
419     std::vector<sf::Vector2u> storagePositions;
420     for (size_t row = 0; row < _height; ++row) {
421         for (size_t col = 0; col < _width; ++col) {
422             if (_board[row][col] == 'a' || _board[row][col] == '1') {
423                 storagePositions.push_back({static_cast<unsigned int>(col)
                    ,
```

```
                    static_cast<unsigned int>(row)});
            }
        }
    }
    return storagePositions;
}

namespace SB {
std::ostream& operator<<(std::ostream& out, const SB::Sokoban& s) {
    out << s._height << " " << s._width << std::endl;
    for (const auto& row : s._board) {
        for (const auto& cell : row) {
            out << cell;
        }
        out << std::endl;
    }
    return out;
}

std::istream& operator>>(std::istream& in, SB::Sokoban& s) {
    in >> s._height >> s._width;
    in.ignore();
    s._board.resize(s._height, std::vector<char>(s._width));

    for (size_t i = 0; i < s._height; ++i) {
        for (size_t j = 0; j < s._width; ++j) {
            in >> s._board[i][j];
        }
    }

    s._initialBoard = s._board;
    s._playerPosition = s.playerLoc();

    return in;
}
}  // namespace SB

test.cpp:
// COPYRIGHT 2025 Kaden Gardiner
#include <fstream>
#include "Sokoban.hpp"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Sokoban
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE(LotsOfBoxesTest) {
    SB::Sokoban s;
    std::ifstream in("sokoban/level5.lvl");
    in >> s;
    in.close();

```

```
476     s.movePlayer(SB::Direction::Up);
477     s.movePlayer(SB::Direction::Up);
478     s.movePlayer(SB::Direction::Up);
479     s.movePlayer(SB::Direction::Up);
480     s.movePlayer(SB::Direction::Right);
481     s.movePlayer(SB::Direction::Right);
482     s.movePlayer(SB::Direction::Right);
483     s.movePlayer(SB::Direction::Right);
484     s.movePlayer(SB::Direction::Down);
485     s.movePlayer(SB::Direction::Right);
486     s.movePlayer(SB::Direction::Up);
487 
488     BOOST_CHECK(s.isWon());
489 }
490 
491 BOOST_AUTO_TEST_CASE(BoxWallTest) {
492     SB::Sokoban s;
493     std::ifstream in("sokoban/level1.lvl");
494     in >> s;
495     in.close();
496 
497     s.movePlayer(SB::Direction::Right);
498     s.movePlayer(SB::Direction::Right);
499     s.movePlayer(SB::Direction::Right);
500     s.movePlayer(SB::Direction::Right);
501     sf::Vector2u playerPos = s.playerLoc();
502     s.movePlayer(SB::Direction::Right);
503     sf::Vector2u newPlayerPos = s.playerLoc();
504 
505     BOOST_CHECK_EQUAL(playerPos.x, newPlayerPos.x);
506     BOOST_CHECK_EQUAL(playerPos.y, newPlayerPos.y);
507 }
508 
509 BOOST_AUTO_TEST_CASE(BoxBoxTest) {
510     SB::Sokoban s;
511     std::ifstream in("sokoban/level5.lvl");
512     in >> s;
513     in.close();
514 
515     s.movePlayer(SB::Direction::Right);
516     s.movePlayer(SB::Direction::Right);
517     s.movePlayer(SB::Direction::Right);
518     s.movePlayer(SB::Direction::Right);
519 
520     s.movePlayer(SB::Direction::Up);
521     s.movePlayer(SB::Direction::Up);
522     s.movePlayer(SB::Direction::Up);
523     s.movePlayer(SB::Direction::Right);
524     s.movePlayer(SB::Direction::Up);
525     s.movePlayer(SB::Direction::Left);
526     s.movePlayer(SB::Direction::Left);
527     sf::Vector2u playerPos = s.playerLoc();
```

```
528     s.movePlayer(SB::Direction::Left);
529     sf::Vector2u newPlayerPos = s.playerLoc();
530 
531     BOOST_CHECK_EQUAL(playerPos.x, newPlayerPos.x);
532     BOOST_CHECK_EQUAL(playerPos.y, newPlayerPos.y);
533 }
534 
535 BOOST_AUTO_TEST_CASE(MoveOffScreenTest) {
536     SB::Sokoban s;
537     std::ifstream in("sokoban/walkover.lvl");
538     in >> s;
539     in.close();
540 
541     BOOST_CHECK_EQUAL(s.playerLoc().x, 2);
542     BOOST_CHECK_EQUAL(s.playerLoc().y, 5);
543     s.movePlayer(SB::Direction::Left);
544     s.movePlayer(SB::Direction::Left);
545     s.movePlayer(SB::Direction::Left);
546     BOOST_CHECK_EQUAL(s.playerLoc().x, 0);
547 }
548 
549 BOOST_AUTO_TEST_CASE(MultipleTargetsTest) {
550     SB::Sokoban s;
551     std::ifstream in("sokoban/level6.lvl");
552     in >> s;
553     in.close();
554 
555     s.movePlayer(SB::Direction::Right);
556     s.movePlayer(SB::Direction::Up);
557     s.movePlayer(SB::Direction::Up);
558     s.movePlayer(SB::Direction::Up);
559     s.movePlayer(SB::Direction::Up);
560     s.movePlayer(SB::Direction::Right);
561     s.movePlayer(SB::Direction::Up);
562     s.movePlayer(SB::Direction::Left);
563     BOOST_CHECK(!s.isWon());
564     s.movePlayer(SB::Direction::Down);
565     s.movePlayer(SB::Direction::Down);
566     s.movePlayer(SB::Direction::Down);
567     s.movePlayer(SB::Direction::Down);
568     s.movePlayer(SB::Direction::Right);
569     s.movePlayer(SB::Direction::Right);
570     s.movePlayer(SB::Direction::Right);
571     s.movePlayer(SB::Direction::Up);
572     s.movePlayer(SB::Direction::Right);
573     s.movePlayer(SB::Direction::Down);
574     BOOST_CHECK(s.isWon());
575 }
576 
577 BOOST_AUTO_TEST_CASE(missingSymbolTest) {
578     SB::Sokoban sokoban;
579     char ch;
```

```
580     bool tof = false;
581     std::ifstream in("sokoban/swapoff.lvl");
582     std::stringstream s;
583     in >> sokoban;
584     in.close();
585     s << sokoban;
586     while (s >> ch) {
587         if (ch == '1') {
588             tof = true;
589         }
590     }
591     BOOST_CHECK(tof);
592 }
```