HELHa
Haute École Louvain en Hainaut

Sokoban

Giorgio Caculli LA196672, Guillaume Lambert LA198116, Tanguy Taminiau LA199566, Nathan Thaon LA188132
Groupe B01

8 décembre 2021

ÉCONOMIQUE

Table des matières

1 Introduction

Dans le cadre du cours de développement de jeux vidéo de l'UE 308, nous avons dû créer un jeu de manière autonome. L'objectif pédagogique de ce projet est de pousser l'élève à mieux appréhender la programmation en C++ vue au cours ainsi que d'apprendre à utiliser une librairie graphique en C++. De plus, la grande liberté accordée à ce projet oblige l'élève à apprendre à se documenter, savoir faire des recherches. L'objectif du projet est simple, créer un jeu selon notre propre envie.

2 Présentation du sujet

Notre jeu sera un jeu de type puzzle. Il sera en vue 2D, vue du haut. Le but du jeu sera d'atteindre un objectif, en se frayant un chemin via la résolution d'un puzzle. La mécanique principale de ce jeu sera de pouvoir pousser une caisse pour nous permettre d'atteindre notre objectif qui sera de pousser ces caisses sur certains points pour terminer le niveau. Les mouvements du personnage et des caisses se feront en case par case et les caisses ne pourront pas être poussées deux par deux. Il y aura aussi la présence d'un compteur de mouvements et un un compteur de reset.

2.1 Spécification technique

2.1.1 GUI : SFML

Figure 1 – Logo SFML

SFML est une librairie qui donne accès à une vaste variété de fonctionnalités purement écrites en C++. Les cinq fonctionnalités dont nous disposons sont les gestions suivantes :

- Toute interaction avec le système d'exploitation
- Fenêtrage
- Graphismes
- Son
- Réseau

SFML permet le cross-platforming : un logiciel codé avec SFML aura le même visuel indépendamment du système d'exploitation sur lequel le jeu tourne.

2.1.2 Librairie Boost

Figure 2 – Logo Boost

La librairie de logging que nous utiliserons se nomme Boost. En quelques mots, la librairie Boost est elle-même un ensemble de librairies permettant d'étendre les fonctionnalités de C++. Dans notre cas, nous utiliserons les

fonctionnalités prédéfinies de Boost. Notamment Log, qui nous donne accès à la possibilité d'enregistrer les différentes interactions qui ont eu lieu lors de l'exécution du jeu.

2.1.3 Fichiers

Les niveaux et les sauvegardes seront stockés dans des fichiers purement textuels. Ces fichiers ne stockeront que le design des niveaux ou de la partie en cours. Comme déclaré précédemment, Boost est un ensemble de librairies, dans cet ensemble il existe la librairie Boost.JSON. Grâce à cette librairie, nous serons capable de stocker des informations en format JSON, comme par exemple, une liste des scores.

2.1.4 OS

Les systèmes d'exploitation sur lesquels nous testerons notre jeu sont les suivants :
— Linux
— MacOS 11
— MS Windows 10

3 Analyse

3.1 Product Backlog

US-01	En tant qu'utilisateur je voudrais reset une partie.
US-02	En tant qu'utilisateur je voudrais mettre en pause la partie.
US-03	En tant qu'utilisateur je voudrais sauvegarder une partie.
US-04	En tant qu'utilisateur je voudrais charger des niveaux personnalisé.
US-05	En tant qu'utilisateur je voudrais arrêter mon jeu à tout môment.
US-06	En tant qu'utilisateur je voudrais une musique de fond.
US-07	En tant qu'utilisateur je voudrais gérer le volume de la musique et des effets.
US-08	En tant qu'utilisateur je voudrais créer des niveaux personnaliser.

4 diagramme

4.1 Diagramme UML

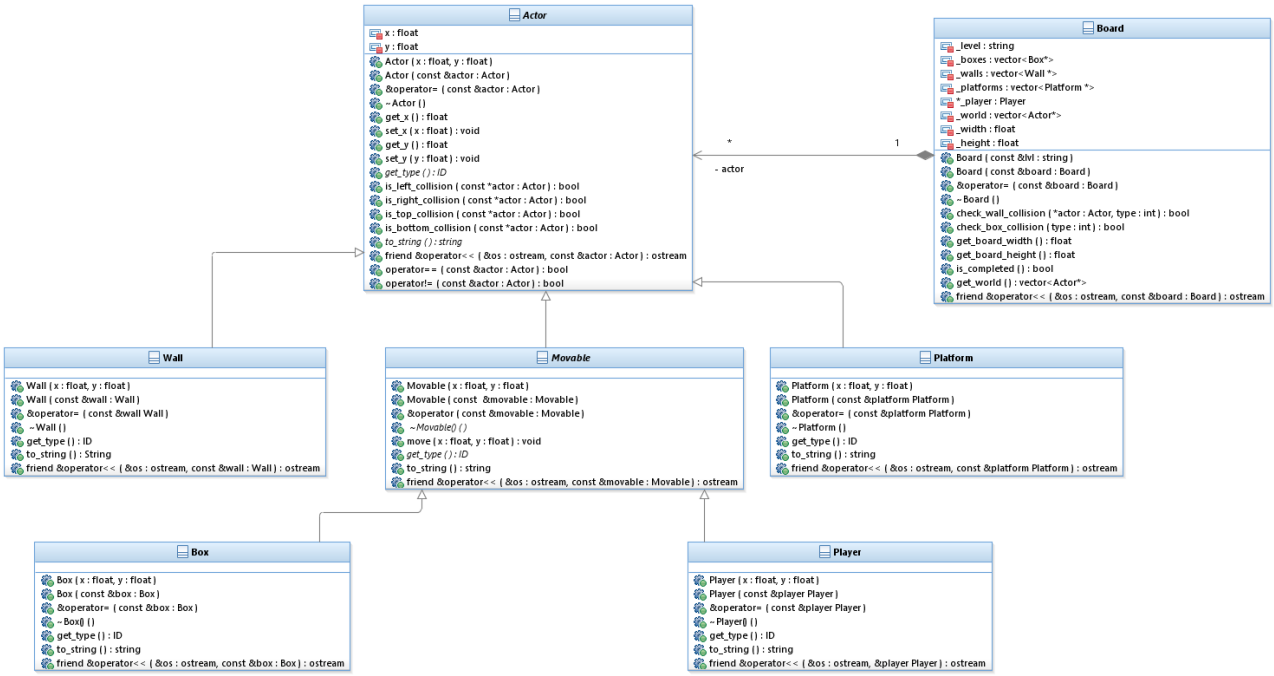

FIGURE 3 – diagramme de classe

La Board est composé d'Actor ou de ses spécialisations. Plus précisement, une board est composé d'objet immobile tels que des Walls et des Platforms mais aussi d'objet mobile tels que des Box et un personnage.

4.2 Diagramme d'activité

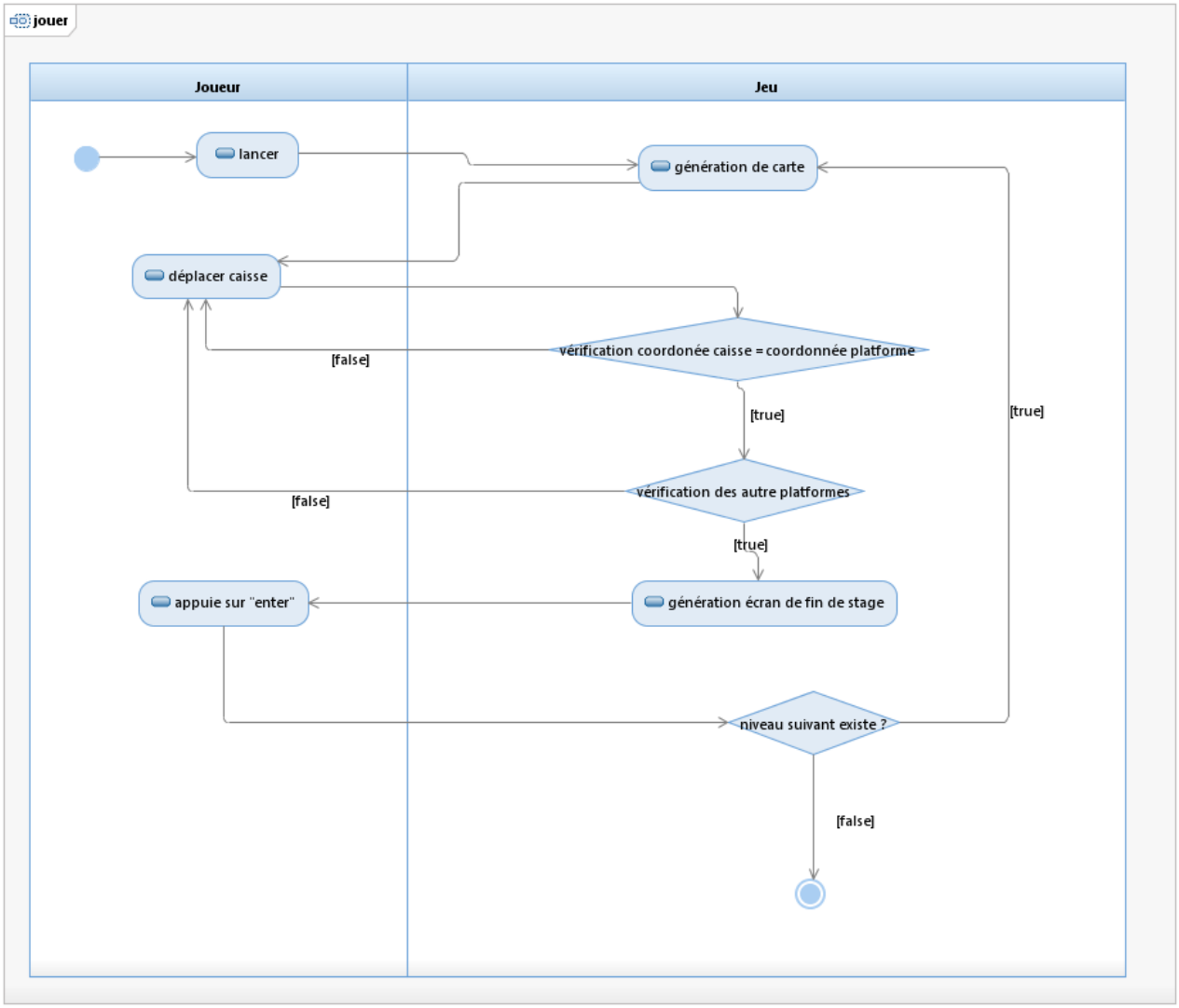

FIGURE 4 – diagramme d'activité

4.3 Diagramme Design Pattern State

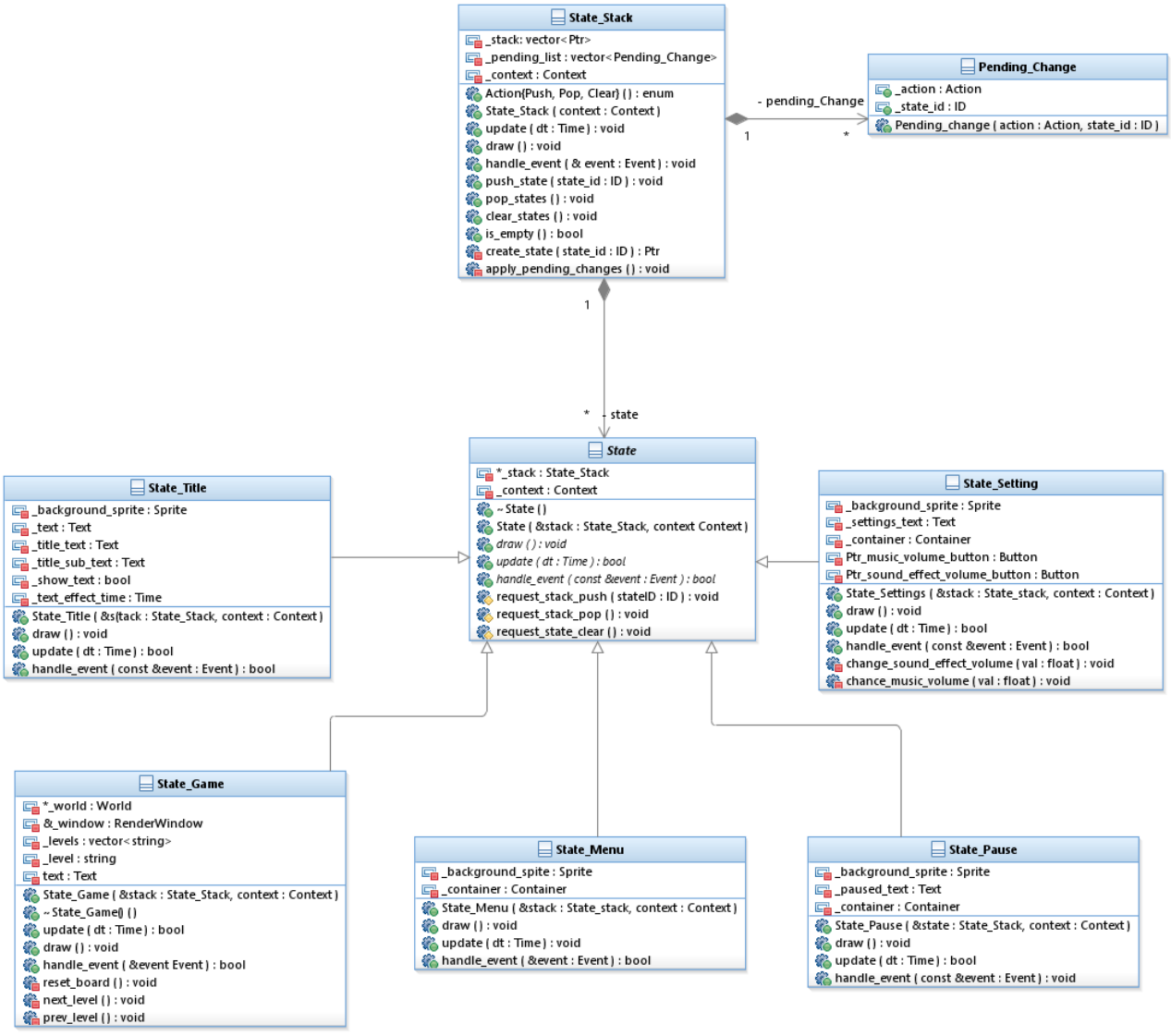

FIGURE 5 – diagramme du design pattern state

5 Implémentation

5.1 Présentation du jeu

Comme dit précédemment, notre jeu est un jeu de type puzzle en 2D en vue du dessus. Notre jeu compte 25 niveaux de bases, il est possible de créer nos propres niveau et de les inclure dans notre liste de niveaux. Pour terminer un niveau (et donc finir le puzzle), il faut mettre toutes les caisses sur une platforme, les platformes ne sont pas spécifiques à une caisse.

5.2 Présentation de l'UI

5.2.1 Ecran titre

FIGURE 6 – écran titre

Voici à quoi ressemble l'écran titre de notre jeu, comme vous pouvez le remarquer, l'écran titre est composé du nom de l'application (écrit en alphabet latin et en japonais). Une image de présentation du jeu est également présente.

5.2.2 Menu principal

FIGURE 7 – menu principal

Notre menu principal est composé de trois bouttons , Le premier permettant de jouer, le deuxième permet d'accéder au menu "paused" et le troisième permet de quitter le jeu. L'image de fond reste la même qu'à l'écran titre.

5.2.3 Menu pause

FIGURE 8 – menu pause

Les deux premiers bouttons permettent de régler le volume de la musique et le volume des effets sonores. Pour modifier les valeurs, sélectionné le bouton correspondant grâce au flèche haut et bas du clavier. Appuyez sur la touche "enter" et appuyez sur la flèche du haut pour monter le volume ou celle du bas pour le descendre. Une fois le modification faite appuyez sur "esc" pour sortir de la sélection du boutton. Le dernier boutton permet de reprendre le jeu où il a été laissé.

5.2.4 Jeu

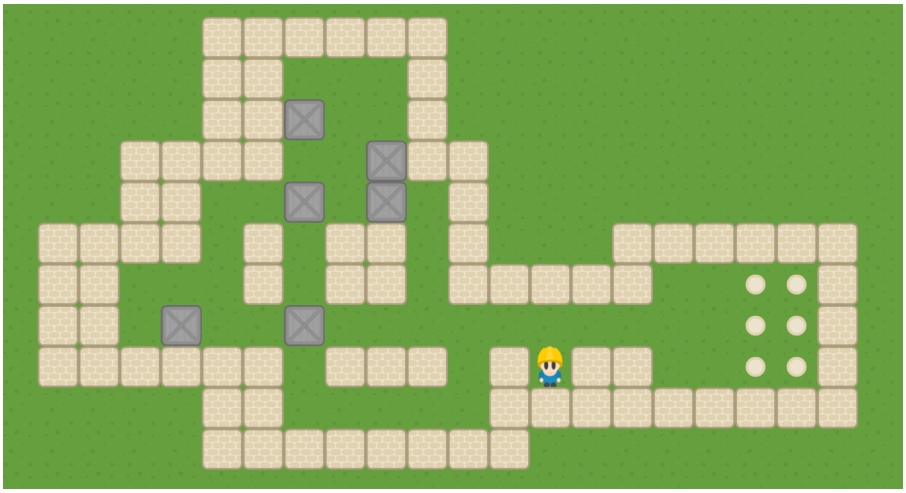

FIGURE 9 – jeu

Voici à quoi ressemble notre jeu. La vue est composé d'un fond à pois (vert sur l'image), de murs qui définissent la limite de la zone du puzzle (beiges sur l'image), de platformes qui définissent l'emplacement de l'objectif (blanches sur l'image) et de caisses (grises sur l'image)qui doivent être déplacées vers les objectifs (blancs sur l'image). les couleurs changent de manière aléatoire pour chaque reset de niveau.

6 Contribution

Giorgio a réalisé l'interface graphique. Etant plus en avance que les autres sur le language C++, il s'est directement attaqué à l'interface graphique donc de la matière qui n'a pas été vue, étant donné que Giorgio avait déjà vu une partie de la matière de C++.

Tanguy a réalisé l'UML de base pour mettre en place le début du projet, celui-ci a été appelé à évoluer avec le développement du projet. Tanguy suite à l'écriture de l'UML a commencé l'écriture des différentes classes du modèle.

Guillaume a réalisé une partie de l'algorithmique du modèle, notamment la gestion des collisions. Guillaume a aussi aidé en naviguant entre les différents participants du projet en fonction de là où il serait le plus utile.

Nathan a réalisé des réflexions sur des design patterns pouvant être utiles à notre jeu et Nathan les a implémentés. Il a aussi beaucoup travaillé sur la documentation pour laisser aux autres le temps de travailler plus sur le projet.

7 Conclusion

Ce projet nous permit de nous rendre compte de ce qu'est un projet de création de jeu vidéo. Cela nous a aussi permit de nous familiariser avec le language de programmation C++ mais aussi le logiciel de version git. De plus, nous avons appris à bien communiquer et s'organiser entre collègues/coéquipiers. Ce projet nous a permis aussi d'apprendre à combiner ce que nous apprenons en cours avec de la recherche de documentation sur internet. En finalité, nous sommes contents du résultat obtenu actuellement, cependant, nous pensons arrivez plus loin et avons légèrement sous estimé le temps nécessaire pour le jeu initialement prévu. Nous tenons à remercier ceux qui nous donné leurs avis par rapport au design du jeu. Et nous tenons à remercier particulièrement Mr. Altares pour ses bons conseils.

8 Bibliographie

Jan Haller, Henrik Vogelius Hansson, Artur Moreira - *SFML Game Development* - Packt - 2013

Milcho G. Milchev - *SFML Essentials* - Packt - 2013

Maxime Barbier - *SFML Blueprints* - Packt - 2015

Raimondas Pupius - *SFML Game Development By Example* - Packt - 2015

Raimondas Pupius - *Mastering SFML Game Development* - Packt - 2017

9 Code source

9.1 Aspect Intéressant

Un aspect intéressant qui a demandé beaucoup de recherché était de réussir à mettre en place la randomisation des différents aspects graphiques présents sur la carte. Afin de rendre le jeu plus varié, différentes algorithmes propres à C++ ont été exploité. Pour créer un niveau de randomisation, C++ demande à mettre en place un générateur de « seed » qui permettra par la suite de choisir de manière aléatoire entre les différentes choix proposés dans les énumérations des couleurs. Pour ce faire, on a exploité le « mersenne twister engine », aussi connu en C++ sous le nom de « mt19937 ». Ce mécanisme permet de retourner un entier de la taille maximale de 32 bit, en partant d'un seed randomiquement généré. C'est grâce à toutes ses engine qu'on a su avoir une complète randomisation des assets lors de l'affichage de la carte.

9.2 Main

9.2.1 main.hpp

```hpp
#ifndef SOKOBAN_CPP_MAIN_HPP
#define SOKOBAN_CPP_MAIN_HPP

/**
 * Default definition of the main function
 *
 * @param argc The number of arguments
 * @param argv Array containing the argument
 * @return 0 if the program executed correctly
 */
int main( int argc, char *argv[] );

#endif //SOKOBAN_CPP_MAIN_HPP
```

9.2.2 main.cpp

```cpp
#include "main.hpp"

#include "ui/Menu.hpp"
#include "util/Logger.hpp"

using namespace sokoban::ui;
using namespace sokoban::util;

/**
 * Main function that initializes the game's logger.
 * The logger will log everything inside a file called sokoban.log
 * File which is instantiated upon execution of the program.
 * Once the logger is initialized, we initialize the menu
 * which will be used to start the application.
 * If no exception is caught, the program will return 0, otherwise -1.
 */
int main( int argc, char *argv[] )
{
    Logger logger( "sokoban.log" );
    remove( logger.get_file_name().c_str() );

    Logger::log( LoggerLevel::INFO, "Starting Menu..." );

    Menu menu;

    Logger::log( LoggerLevel::INFO, "Running Game..." );

    int res = menu.launch_application();

    Logger::log( LoggerLevel::INFO, "Closing Sokoban..." );

    return res;
}
```

9.3 Model

9.3.1 Actor.hpp

```hpp
#ifndef SOKOBAN_ACTOR_HPP
#define SOKOBAN_ACTOR_HPP

#include <array>
#include <string>
#include <ostream>

namespace sokoban
{
    namespace model
    {
        /**
         * Actor class
         * Parent class for any actor that will be displayed throughout the game
         * Parent class of both movable and non movable actors
         */
        class Actor
        {
        private:
            float _x; /** The X coordinate on the board */
            float _y; /** The Y coordinate on the board */
        public:
            enum ID /** We consider there to be 4 types of actors */
```

```
24              {
25                  BOX, /** A box */
26                  PLATFORM, /** A platform on which we put the box */
27                  PLAYER, /** A player */
28                  WALL /** A wall */
29              };
30              Actor( float x, float y );
31              Actor( const Actor &actor );
32              Actor &operator=( const Actor &actor );
33              virtual ~Actor() = default;
34              float get_x() const;
35              void set_x( float x );
36              float get_y() const;
37              void set_y( float y );
38              virtual ID get_type() const = 0;
39              bool is_left_collision( const Actor *actor ) const;
40              bool is_right_collision( const Actor *actor ) const;
41              bool is_top_collision( const Actor *actor ) const;
42              bool is_bottom_collision( const Actor *actor ) const;
43              virtual std::string to_string() const;
44              friend std::ostream &operator<<( std::ostream &os, const Actor &actor );
45              bool operator==( const Actor &actor ) const;
46              bool operator!=( const Actor &actor ) const;
47          };
48      }
49  }
50
51  #endif //SOKOBAN_ACTOR_HPP
```

9.3.2 Actor.cpp

```
1   #include "Actor.hpp"
2
3   #include <sstream>
4
5   using namespace sokoban::model;
6
7   const float SPACE = 64.f; /** Dans ce cas, les images mesurent 64x64 */
8
9   /**
10   * This constructor will initialise the different pieces of informations which characterise an actor
11   * @param x The X coordinate.
12   * @param y The Y coordinate.
13   * @param asset_coords The coordinates on the sprite sheet which represent it graphically.
14   */
15  Actor::Actor( float x, float y )
16          : _x( x )
17            , _y( y )
18  {
19  }
20
21  /**
22   * Copy Constructor
23   * @param actor The actor from which we wish to create a copy
24   */
25  Actor::Actor( const Actor &actor )
26          : Actor( actor._x, actor._y )
27  {
28  }
29
30  /**
31   * Redefinition of the "=" operator
32   * @param actor The actor from which we wish to copy the information
33   * @return The new instance of an actor with the same information as the one on the right
34   */
35  Actor &Actor::operator=( const Actor &actor )
36  {
37      if ( &actor != this )
38      {
39          _x = actor._x;
40          _y = actor._y;
41      }
42      return *this;
43  }
44
45  /**
46   * X coordinate getter
47   * @return The value of X
48   */
49  float Actor::get_x() const
50  {
51      return _x;
52  }
```

```
53
54 /**
55  * X coordinate setter
56  * @param x The new value of X
57  */
58 void Actor::set_x( float x )
59 {
60     this->_x = x;
61 }
62
63 /**
64  * Y coordinate getter
65  * @return The value of Y
66  */
67 float Actor::get_y() const
68 {
69     return _y;
70 }
71
72 /**
73  * Y coordinate setter
74  * @param y The new value of Y
75  */
76 void Actor::set_y( float y )
77 {
78     this->_y = y;
79 }
80
81 /**
82  * Function whose purpose is to verify if a left collision took place between two actors.
83  * A movable actor will call this function and will check the following cases:
84  * Let's assume Actor 1's X coordinates is x1 and Actor 2's X coordinates is x2:
85  * Let's assume Actor 1's Y coordinates is y1 and Actor 2's Y coordinates is y2:
86  * SPACE represents the amount of space between each object before the collision:
87  * If x1 - SPACE = x2 and Y1 = Y2
88  * Then a collision on the left took place.
89  * @param actor The actor on the left
90  * @return true if the collision took place, false otherwise
91  */
92 bool Actor::is_left_collision( const Actor *actor ) const
93 {
94     return get_x() - SPACE == actor->get_x() && get_y() == actor->get_y();
95 }
96
97 /**
98  * Function whose purpose is to verify if a right collision took place between two actors.
99  * A movable actor will call this function and will check the following cases:
100  * Let's assume Actor 1's X coordinates is x1 and Actor 2's X coordinates is x2:
101  * Let's assume Actor 1's Y coordinates is y1 and Actor 2's Y coordinates is y2:
102  * SPACE represents the amount of space between each object before the collision:
103  * If x1 + SPACE = x2 and y1 = y2
104  * Then a collision on the right took place.
105  * @param actor The actor on the right
106  * @return true if the collision took place, false otherwise
107  */
108 bool Actor::is_right_collision( const Actor *actor ) const
109 {
110     return get_x() + SPACE == actor->get_x() && get_y() == actor->get_y();
111 }
112
113 /**
114  * Function whose purpose is to verify if a top collision took place between two actors.
115  * A movable actor will call this function and will check the following cases:
116  * Let's assume Actor 1's X coordinates is x1 and Actor 2's X coordinates is x2:
117  * Let's assume Actor 1's Y coordinates is y1 and Actor 2's Y coordinates is y2:
118  * SPACE represents the amount of space between each object before the collision:
119  * If y1 - SPACE = y2 and x1 = x2
120  * Then a collision on the top took place.
121  * @param actor The actor on the top
122  * @return true if the collision took place, false otherwise
123  */
124 bool Actor::is_top_collision( const Actor *actor ) const
125 {
126     return get_y() - SPACE == actor->get_y() && get_x() == actor->get_x();
127 }
128
129 /**
130  * Function whose purpose is to verify if a bottom collision took place between two actors.
131  * A movable actor will call this function and will check the following cases:
132  * Let's assume Actor 1's X coordinates is x1 and Actor 2's X coordinates is x2:
133  * Let's assume Actor 1's Y coordinates is y1 and Actor 2's Y coordinates is y2:
134  * SPACE represents the amount of space between each object before the collision:
135  * If y1 + SPACE = y2 and x1 = x2
136  * Then a collision on the bottom took place.
```

```
137  * @param actor The actor on the bottom
138  * @return true if the collision took place, false otherwise
139  */
140 bool Actor::is_bottom_collision( const Actor *actor ) const
141 {
142     return get_y() + SPACE == actor->get_y() && get_x() == actor->get_x();
143 }
144
145 /**
146  * Textual output for the actor's coordinates
147  * @return X: <the value of the x coordinate> Y: <the value of the y coordinate>
148  */
149 std::string Actor::to_string() const
150 {
151     std::stringstream ss;
152     ss << "X: " << _x << " Y: " << _y;
153     return ss.str();
154 }
155
156 /**
157  * Redefinition of the << operator
158  * This redefinition will allow us to directly output the object without
159  * having to call the to_string() function within the << operators
160  * @return The textual output from the function to_string()
161  */
162 std::ostream &sokoban::model::operator<<( std::ostream &os, const Actor &actor )
163 {
164     os << actor.to_string();
165     return os;
166 }
167
168 /**
169  * Redefinition of the == operator
170  * This will allow us to check whether or not two actors are exactly the same.
171  * @return true if same actor, false if not
172  */
173 bool Actor::operator==( const Actor &actor ) const
174 {
175     return _x == actor._x &&
176            _y == actor._y;
177 }
178
179 /**
180  * Redefinition of the != operator
181  * This will allow us to check whether two actors are not the same
182  * @return the opposite of the == operator
183  */
184 bool Actor::operator!=( const Actor &actor ) const
185 {
186     return !( actor == *this );
187 }
```

9.3.3 Movable.hpp

```
1  #ifndef SOKOBAN_MOVABLE_HPP
2  #define SOKOBAN_MOVABLE_HPP
3
4  #include "Actor.hpp"
5
6  #include <array>
7  #include <string>
8  #include <ostream>
9
10 /**
11  * Class that handles actors capable of moving
12  * This class inherits from Actor
13  */
14 namespace sokoban
15 {
16     namespace model
17     {
18         class Movable
19                 : public Actor
20         {
21         public:
22             Movable( float x, float y );
23             Movable( const Movable &movable );
24             Movable &operator=( const Movable &movable );
25             ~Movable() override = default;
26             void move( float x, float y );
27             ID get_type() const override = 0;
28             std::string to_string() const override;
29             friend std::ostream &operator<<( std::ostream &os, const Movable &movable );
```

```
30          };
31      }
32  }
33
34  #endif //SOKOBAN_MOVABLE_HPP
```

9.3.4 Movable.cpp

```
1   #include "Movable.hpp"
2
3   #include <sstream>
4   #include <utility>
5
6   using namespace sokoban::model;
7
8   /**
9    * Just like the Actor class, the object 'movable' must also instance the X and Y coordinates
10   */
11  Movable::Movable( float x, float y )
12          : Actor( x, y )
13  {
14  }
15
16  /**
17   * Copy constructor for the movable actor
18   */
19  Movable::Movable( const Movable &movable )
20          : Movable( movable.get_x(), movable.get_y() )
21  {
22  }
23
24  /**
25   * Redefinition of the = operator
26   * @param movable The movable object we wish to copy the information from
27   * @return The new instance of the actor with the copied information
28   */
29  Movable &Movable::operator=( const Movable &movable )
30  {
31      if ( &movable != this )
32      {
33          set_x( movable.get_x() );
34          set_y( movable.get_y() );
35      }
36      return *this;
37  }
38
39  /**
40   * Make the object transition from a coordinate to another
41   * @param x The X units to move to
42   * @param y The Y units to move to
43   */
44  void Movable::move( float x, float y )
45  {
46      set_x( get_x() + x );
47      set_y( get_y() + y );
48  }
49
50  /**
51   * Textual output for the actor
52   * @return Display the information related to the movable object
53   */
54  std::string Movable::to_string() const
55  {
56      std::stringstream ss;
57      ss << Actor::to_string();
58      return ss.str();
59  }
60
61  /**
62   * Redefinition of the << operator
63   * @param os The desired output stream
64   * @param movable The movable object to get the information from
65   * @return The textual output
66   */
67  std::ostream &sokoban::model::operator<<( std::ostream &os, const Movable &movable )
68  {
69      os << movable.to_string();
70      return os;
71  }
```

9.3.5 Box.hpp

```hpp
#ifndef SOKOBAN_BOX_HPP
#define SOKOBAN_BOX_HPP

#include "Movable.hpp"

#include <map>
#include <array>
#include <string>
#include <ostream>

namespace sokoban
{
    namespace model
    {
        /**
         * Box class
         * This class is an actor that can move, as such it inherits from the classe Movable
         */
        class Box
                : public Movable
        {
        public:
            Box( float x, float y );
            Box( const Box &box );
            Box &operator=( const Box &box );
            ~Box() override;
            ID get_type() const override;
            std::string to_string() const override;
            friend std::ostream &operator<<( std::ostream &os, const Box &box );
        };
    }
}

#endif //SOKOBAN_BOX_HPP
```

9.3.6 Box.cpp

```cpp
#include "Box.hpp"

#include "../util/Logger.hpp"

#include <sstream>
#include <iostream>

using namespace sokoban::model;
using namespace sokoban::util;

Box::Box( float x, float y )
        : Movable( x, y )
{
}

Box::Box( const Box &box )
        : Box( box.get_x(), box.get_y() )
{
}

Box &Box::operator=( const Box &box )
{
    if ( &box != this )
    {
        set_x( box.get_x() );
        set_y( box.get_y() );
    }
    return *this;
}

Box::~Box()
{
    Logger::log( LoggerLevel::INFO, "Deletion " + Box::to_string() );
}

Actor::ID Box::get_type() const
{
    return Actor::BOX;
}

std::string Box::to_string() const
{
    std::stringstream ss;
    ss << "Box: " << Movable::to_string();
    return ss.str();
}
```

```
47
48 std::ostream &sokoban::model::operator<<( std::ostream &os, const Box &box )
49 {
50     os << box.to_string();
51     return os;
52 }
```

9.3.7 Platform.hpp

```
1  #ifndef SOKOBAN_PLATFORM_HPP
2  #define SOKOBAN_PLATFORM_HPP
3
4  #include "Actor.hpp"
5
6  #include <map>
7  #include <array>
8  #include <string>
9  #include <ostream>
10
11 namespace sokoban
12 {
13     namespace model
14     {
15         class Platform
16                 : public Actor
17         {
18         public:
19             Platform( float x, float y );
20             Platform( const Platform &platform );
21             Platform &operator=( const Platform &platform );
22             ~Platform() override;
23             ID get_type() const override;
24             std::string to_string() const override;
25             friend std::ostream &operator<<( std::ostream &os, const Platform &platform );
26         };
27     }
28 }
29
30 #endif //SOKOBAN_PLATFORM_HPP
```

9.3.8 Platform.cpp

```
1  #include "Platform.hpp"
2
3  #include "../util/Logger.hpp"
4
5  #include <sstream>
6  #include <iostream>
7
8  using namespace sokoban::model;
9  using namespace sokoban::util;
10
11 /**
12  * Default constructor for the platforms
13  * @param x The X coordinates on the board
14  * @param y The Y coordinates on the board
15  */
16 Platform::Platform( float x, float y )
17         : Actor( x, y )
18 {
19 }
20
21 /**
22  * Copy constructor for the platform actor
23  * @param platform The platform we wish to copy the information from
24  */
25 Platform::Platform( const Platform &platform )
26         : Platform( platform.get_x(), platform.get_y() )
27 {
28 }
29
30 /**
31  * Redefinition of the = operator
32  * @param platform The platform we wish to copy the information from
33  * @return New instance of a platform with the copied information
34  */
35 Platform &Platform::operator=( const Platform &platform )
36 {
37     if ( &platform != this )
38     {
39         set_x( platform.get_x() );
40         set_y( platform.get_y() );
```

```
41      }
42      return *this;
43  }
44
45  /**
46   * Default destructor for the platforms
47   */
48  Platform::~Platform()
49  {
50      Logger::log( LoggerLevel::INFO, "Deletion " + Platform::to_string() );
51  }
52
53  /**
54   * Getter meant to retrieve the nature of a platform actor
55   * @return The fact that the actor is actually a platform
56   */
57  Actor::ID Platform::get_type() const
58  {
59      return Actor::PLATFORM;
60  }
61
62  /**
63   * Textual output containing the platform's information
64   * @return Text containing the info
65   */
66  std::string Platform::to_string() const
67  {
68      std::stringstream ss;
69      ss << "Platform: " << Actor::to_string();
70      return ss.str();
71  }
72
73  /**
74   * Redefinition of the << operator
75   * @param os The desired output stream
76   * @param platform The platform we wish to output
77   * @return Textual output displaying the platform's information
78   */
79  std::ostream &sokoban::model::operator<<( std::ostream &os, const Platform &platform )
80  {
81      os << platform.to_string();
82      return os;
83  }
```

9.3.9 Player.hpp

```
1  #ifndef SOKOBAN_PLAYER_HPP
2  #define SOKOBAN_PLAYER_HPP
3
4  #include "Movable.hpp"
5
6  #include <map>
7  #include <array>
8  #include <string>
9  #include <ostream>
10
11 namespace sokoban
12 {
13     namespace model
14     {
15         class Player
16                 : public Movable
17         {
18         public:
19             Player( float x, float y );
20             Player( const Player &player );
21             Player &operator=( const Player &player );
22             ~Player() override;
23             ID get_type() const override;
24             std::string to_string() const override;
25             friend std::ostream &operator<<( std::ostream &os, const Player &player );
26         };
27     }
28 }
29
30 #endif //SOKOBAN_USER_HPP
```

9.3.10 Player.cpp

```
1  #include "Player.hpp"
2
3  #include "../util/Logger.hpp"
```

```
4
5  #include <sstream>
6  #include <iostream>
7
8  using namespace sokoban::model;
9  using namespace sokoban::util;
10
11 /**
12  * Default constructor for the player
13  * @param x The X coordinates on the board
14  * @param y The Y coordinates on the board
15  */
16 Player::Player( float x, float y )
17         : Movable( x, y )
18 {
19 }
20
21 /**
22  * Copy constructor for the player
23  * @param player The player we wish to copy the information from
24  */
25 Player::Player( const Player &player )
26         : Player( player.get_x(), player.get_y() )
27 {
28 }
29
30 /**
31  * Redefinition of the = operator
32  * @param player The player we wish to copy the information from
33  * @return The new instance of a player with the copied information
34  */
35 Player &Player::operator=( const Player &player )
36 {
37     if ( &player != this )
38     {
39         set_x( player.get_x() );
40         set_y( player.get_y() );
41     }
42     return *this;
43 }
44
45 /**
46  * Default destructor for the player
47  */
48 Player::~Player()
49 {
50     Logger::log( LoggerLevel::INFO, "Deletion " + Player::to_string() );
51 }
52
53 /**
54  * Getter used to retrieve a player actor's nature
55  * @return The fact that the actor is a player
56  */
57 Actor::ID Player::get_type() const
58 {
59     return Actor::PLAYER;
60 }
61
62 /**
63  * Textual output containing the Player's information
64  * @return Text with the player's info
65  */
66 std::string Player::to_string() const
67 {
68     std::stringstream ss;
69     ss << "Player: " << Movable::to_string();
70     return ss.str();
71 }
72
73 /**
74  * Redefinition of the << operator
75  * @param os The desired output stream
76  * @param player The player we wish to output
77  * @return Textual output containing the player's information
78  */
79 std::ostream &sokoban::model::operator<<( std::ostream &os, const Player &player )
80 {
81     os << player.to_string();
82     return os;
83 }
```

9.3.11 Wall.hpp

```hpp
#ifndef SOKOBAN_WALL_HPP
#define SOKOBAN_WALL_HPP

#include "Actor.hpp"

#include <map>
#include <array>
#include <string>
#include <ostream>

namespace sokoban
{
    namespace model
    {
        class Wall
                : public Actor
        {
        public:
            Wall( float x, float y );
            Wall( const Wall &wall );
            Wall &operator=( const Wall &wall );
            ~Wall() override;
            ID get_type() const override;
            std::string to_string() const override;
            friend std::ostream &operator<<( std::ostream &os, const Wall &wall );
        };
    }
}

#endif //SOKOBAN_WALL_HPP
```

9.3.12 Wall.cpp

```cpp
#include "Wall.hpp"

#include "../util/Logger.hpp"

#include <sstream>
#include <iostream>

using namespace sokoban::model;
using namespace sokoban::util;

/**
 * Default constructor for the walls
 * @param x The X coordinate on the board
 * @param y The Y coordinate on the board
 */
Wall::Wall( float x, float y )
        : Actor( x, y )
{
}

/**
 * Copy constructor for the wall
 * @param wall The wall we wish to copy the information from
 */
Wall::Wall( const Wall &wall )
        : Wall( wall.get_x(), wall.get_y() )
{
}

/**
 * Redefinition of the = operator
 * @param wall The wall we wish to copy the information from
 * @return The new instance of wall with the copied information
 */
Wall &Wall::operator=( const Wall &wall )
{
    if ( &wall != this )
    {
        set_x( wall.get_x() );
        set_y( wall.get_y() );
    }
    return *this;
}

/**
 * Default destructor for the wall actor
 */
Wall::~Wall()
{
    Logger::log( LoggerLevel::INFO, "Deletion " + Wall::to_string() );
```

```
}

/**
 * Get the nature of a wall
 * @return The fact that it is a wall actor
 */
Actor::ID Wall::get_type() const
{
    return Actor::WALL;
}

/**
 * Textual output stating the wall's information
 * @return The textual output with the wall's information
 */
std::string Wall::to_string() const
{
    std::stringstream ss;
    ss << "Wall: " << Actor::to_string();
    return ss.str();
}

/**
 * Redefinition of the << operator
 * @param os The desired output stream
 * @param wall The wall we wish to output
 * @return Textual output containing the wall's information
 */
std::ostream &sokoban::model::operator<<( std::ostream &os, const Wall &wall )
{
    os << wall.to_string();
    return os;
}
```

9.3.13 Board.hpp

```
#ifndef SOKOBAN_BOARD_HPP
#define SOKOBAN_BOARD_HPP

#include "Actor.hpp"
#include "Box.hpp"
#include "Wall.hpp"
#include "Player.hpp"
#include "Platform.hpp"

#include <vector>
#include <ostream>

namespace sokoban
{
    namespace model
    {
        /**
         * Board class
         * Meant to build the layout of the leve that the player will be playing
         */
        class Board
        {
        private:
            std::string _level; /** The skeleton of the level */
            std::vector< Box * > _boxes; /** All the boxes in the level */
            std::vector< Wall * > _walls; /** All the walls in the game */
            std::vector< Platform * > _platforms; /** All the platforms in the game */
            Player *_player; /** The main character of the game */
            std::vector< Actor * > _world; /** Every actor in the game */
            float _width; /** The board's maximum width */
            float _height; /** The board's maximum height */
            void init_board();
            void init_world();
            void build_world();
        public:
            enum
            {
                LEFT_COLLISION,
                RIGHT_COLLISION,
                TOP_COLLISION,
                BOTTOM_COLLISION
            };
            explicit Board( const std::string &lvl );
            Board( const Board &board );
            Board &operator=( const Board &board );
            ~Board();
            bool check_wall_collision( Actor *actor, int type );
```

```
48              bool check_box_collision( int type );
49              float get_board_width() const;
50              float get_board_height() const;
51              bool is_completed() const;
52              std::vector< Actor * > get_world();
53              friend std::ostream &operator<<( std::ostream &os, const Board &board );
54          };
55      }
56  }
57
58  #endif //SOKOBAN_BOARD_HPP
```

9.3.14 Board.cpp

```
1   #include "Board.hpp"
2
3   #include <string>
4   #include <sstream>
5   #include <iostream>
6   #include <fstream>
7
8   using namespace sokoban::model;
9
10  const int OFFSET = 64; /** Variable used for the amount of pixels an actor can move */
11  const int SPACE = 64; /** Variable used for the size of each character */
12
13  /**
14   * Constructor for the board
15   * The parameter lvl is meant to be a string containing the path to a text file
16   * That text file will contain the skeleton of the level, meaning that from the
17   * symbols present on the text file, we can build every actor of the game.
18   * # represents a wall
19   * @ represents the player
20   * . represents a platform
21   * $ represents a box
22   * An empty space is just emptiness
23   * @param lvl The path for the level
24   */
25  Board::Board( const std::string &lvl )
26          : _boxes()
27            , _walls()
28            , _platforms()
29            , _player( nullptr )
30            , _world()
31            , _width( 0 )
32            , _height( 0 )
33  {
34      std::string line;
35      std::stringstream ss;
36      std::ifstream level_file( lvl );
37      if ( level_file.is_open() )
38      {
39          while ( std::getline( level_file, line ) )
40          {
41              ss << line << '\n';
42          }
43          level_file.close();
44      }
45      _level = std::move( ss.str() );
46      init_board();
47  }
48
49  /**
50   * Copy constructor of the board
51   * Primarily used to create a backup of the board.
52   * An example of usage for this constructor could be to load a board when
53   * the player hits the reset button
54   * @param board The board from which we wish to create a copy
55   */
56  Board::Board( const Board &board )
57          : Board( board._level )
58  {
59  }
60
61  /**
62   * Redefinition of the = operator
63   * Same usage as the copy constructor
64   * @param board
65   * @return
66   */
67  Board &Board::operator=( const Board &board )
68  {
69      if ( &board != this )
```

```
70      {
71          _level = board._level;
72          _boxes = board._boxes;
73          _walls = board._walls;
74          _platforms = board._platforms;
75          _player = board._player;
76          _world = board._world;
77          _width = board._width;
78          _height = board._height;
79      }
80      init_board();
81      return *this;
82  }
83
84  /**
85   * Board's destructor
86   * Meant to clear the memory from any actor that has been initialised
87   */
88  Board::~Board()
89  {
90      for ( Wall *wall: _walls )
91      {
92          delete wall;
93      }
94      for ( Platform *platform: _platforms )
95      {
96          delete platform;
97      }
98      for ( Box *box: _boxes )
99      {
100         delete box;
101     }
102     delete _player;
103     _boxes.clear();
104     _walls.clear();
105     _platforms.clear();
106     _world.clear();
107 }
108
109 /**
110  * Executions before the world initialization
111  */
112 void Board::init_board()
113 {
114     init_world();
115 }
116
117 /**
118  * World initialization
119  * Meant to initialise every actor on the board
120  * As stated in the constructor:
121  * * # represents a wall
122  * @ represents the player
123  * . represents a platform
124  * $ represents a box
125  * An empty space is just emptiness
126  * Each actor is inserted into its respective vector
127  */
128 void Board::init_world()
129 {
130     _boxes = std::vector< Box * >();
131     _walls = std::vector< Wall * >();
132     _platforms = std::vector< Platform * >();
133     float x = OFFSET;
134     float y = OFFSET;
135
136     Box *box;
137     Wall *wall;
138     Platform *platform;
139     for ( char item: _level )
140     {
141         switch ( item )
142         {
143             case '\n':
144                 y += SPACE;
145                 if ( _width < x )
146                 {
147                     _width = x;
148                 }
149                 x = OFFSET;
150                 break;
151             case '#':
152                 wall = new Wall( x, y );
153                 _walls.insert( _walls.begin(), wall );
```

```
154                 x += SPACE;
155                 break;
156             case '$':
157                 box = new Box( x, y );
158                 _boxes.insert( _boxes.begin(), box );
159                 x += SPACE;
160                 break;
161             case '.':
162                 platform = new Platform( x, y );
163                 _platforms.insert( _platforms.begin(), platform );
164                 x += SPACE;
165                 break;
166             case '@':
167                 _player = new Player( x, y );
168                 x += SPACE;
169                 break;
170             case '*':
171                 box = new Box( x, y );
172                 platform = new Platform( x, y );
173                 _boxes.insert( _boxes.begin(), box );
174                 _platforms.insert( _platforms.begin(), platform );
175                 x += SPACE;
176                 break;
177             case '+':
178                 _player = new Player( x, y );
179                 platform = new Platform( x, y );
180                 _platforms.insert( _platforms.begin(), platform );
181                 x += SPACE;
182                 break;
183             case ' ':
184                 x += SPACE;
185                 break;
186             default:
187                 break;
188         }
189         _height = y;
190     }
191     build_world();
192 }
193
194 /**
195  * Definition of each actor in their respective category.
196  * This will allow is to easily clear out their memory simultaneously
197  */
198 void Board::build_world()
199 {
200     _world = std::vector< Actor * >();
201     for ( Wall *wall: _walls )
202     {
203         _world.insert( _world.begin(), wall );
204     }
205     for ( Platform *platform: _platforms )
206     {
207         _world.insert( _world.begin(), platform );
208     }
209     for ( Box *box: _boxes )
210     {
211         _world.insert( _world.begin(), box );
212     }
213     _world.insert( _world.begin(), _player );
214 }
215
216 /**
217  * Function to check whether there is a collision between an actor and a wall
218  * @param actor The actor causing the collision
219  * @param type The type of collision: TOP, BOTTOM, LEFT, RIGHT
220  * @return true if a collision took place, false if not
221  */
222 bool Board::check_wall_collision( Actor *actor, int type )
223 {
224     switch ( type )
225     {
226         case LEFT_COLLISION:
227             for ( const Wall *wall: _walls )
228             {
229                 if ( actor->is_left_collision( wall ) )
230                 {
231                     return true;
232                 }
233             }
234             return false;
235         case RIGHT_COLLISION:
236             for ( const Wall *wall: _walls )
237             {
```

```
238                 if ( actor->is_right_collision( wall ) )
239                 {
240                     return true;
241                 }
242             }
243             return false;
244         case TOP_COLLISION:
245             for ( const Wall *wall: _walls )
246             {
247                 if ( actor->is_top_collision( wall ) )
248                 {
249                     return true;
250                 }
251             }
252             return false;
253         case BOTTOM_COLLISION:
254             for ( const Wall *wall: _walls )
255             {
256                 if ( actor->is_bottom_collision( wall ) )
257                 {
258                     return true;
259                 }
260             }
261             return false;
262         default:
263             break;
264     }
265     return false;
266 }
267
268 /**
269  * Functions that checks whether there is a collision between an actor and a box
270  * @param type The type of collision: TOP, BOTTOM, LEFT, RIGHT
271  * @return true if a collision took place, false if not
272  */
273 bool Board::check_box_collision( int type )
274 {
275     switch ( type )
276     {
277         case LEFT_COLLISION:
278             for ( Box *box: _boxes )
279             {
280                 if ( _player->is_left_collision( box ) )
281                 {
282                     for ( Box *item: _boxes )
283                     {
284                         if ( box != item )
285                         {
286                             if ( box->is_left_collision( item ) )
287                             {
288                                 return true;
289                             }
290                         }
291                         if ( check_wall_collision( box, LEFT_COLLISION ) )
292                         {
293                             return true;
294                         }
295                     }
296                     box->move( -SPACE, 0 );
297                     is_completed();
298                 }
299             }
300             return false;
301         case RIGHT_COLLISION:
302             for ( Box *box: _boxes )
303             {
304                 if ( _player->is_right_collision( box ) )
305                 {
306                     for ( Box *item: _boxes )
307                     {
308                         if ( box != item )
309                         {
310                             if ( box->is_right_collision( item ) )
311                             {
312                                 return true;
313                             }
314                         }
315                         if ( check_wall_collision( box, RIGHT_COLLISION ) )
316                         {
317                             return true;
318                         }
319                     }
320                     box->move( SPACE, 0 );
321                     is_completed();
```

```
322                 }
323             }
324             return false;
325         case TOP_COLLISION:
326             for ( Box *box: _boxes )
327             {
328                 if ( _player->is_top_collision( box ) )
329                 {
330                     for ( Box *item: _boxes )
331                     {
332                         if ( box != item )
333                         {
334                             if ( box->is_top_collision( item ) )
335                             {
336                                 return true;
337                             }
338                         }
339                         if ( check_wall_collision( box, TOP_COLLISION ) )
340                         {
341                             return true;
342                         }
343                     }
344                     box->move( 0, -SPACE );
345                     is_completed();
346                 }
347             }
348             return false;
349         case BOTTOM_COLLISION:
350             for ( Box *box: _boxes )
351             {
352                 if ( _player->is_bottom_collision( box ) )
353                 {
354                     for ( Box *item: _boxes )
355                     {
356                         if ( box != item )
357                         {
358                             if ( box->is_bottom_collision( item ) )
359                             {
360                                 return true;
361                             }
362                         }
363                         if ( check_wall_collision( box, BOTTOM_COLLISION ) )
364                         {
365                             return true;
366                         }
367                     }
368                     box->move( 0, SPACE );
369                     is_completed();
370                 }
371             }
372             return false;
373         default:
374             break;
375     }
376     return false;
377 }
378
379 /**
380  * Getter for the board width
381  * @return The width of the board
382  */
383 float Board::get_board_width() const
384 {
385     return _width;
386 }
387
388 /**
389  * Getter for the board height
390  * @return The height of the board
391  */
392 float Board::get_board_height() const
393 {
394     return _height;
395 }
396
397 /**
398  * Function that allows the played to know whether the level is completed
399  * @return True if all the boxes are on the platforms
400  */
401 bool Board::is_completed() const
402 {
403     unsigned long number_of_boxes = _boxes.size();
404     int finished_boxes = 0;
405
```

```
406     for ( Box *box: _boxes )
407     {
408         for ( Platform *platform: _platforms )
409         {
410             if ( box->get_x() == platform->get_x() && box->get_y() == platform->get_y() )
411             {
412                 finished_boxes += 1;
413             }
414         }
415     }
416 
417     if ( finished_boxes == number_of_boxes )
418     {
419         return true;
420     }
421 
422     return false;
423 }
424 
425 /**
426  * Redefinition of the << operator
427  * Meant to be used when outputting the board textually
428  * @param os The base output stream
429  * @param board The board to output
430  * @return Textual output of the various actor's positions
431  */
432 std::ostream &sokoban::model::operator<<( std::ostream &os, const Board &board )
433 {
434     os << board._level;
435     for ( Actor *actor: board._world )
436     {
437         std::cout << *actor << std::endl;
438     }
439     return os;
440 }
441 
442 /**
443  * Function to retrieve all the actors on the board
444  * @return The vector containing the various actors
445  */
446 std::vector< Actor * > Board::get_world()
447 {
448     return _world;
449 }
```

9.4 Util

9.4.1 Logger.hpp

```
1  #ifndef SOKOBAN_LOGGER_HPP
2  #define SOKOBAN_LOGGER_HPP
3  
4  #include <string>
5  
6  namespace sokoban
7  {
8      namespace util
9      {
10         enum LoggerLevel
11         {
12             TRACE, /** The lowest level of importance leaving nothing but the existance of the information
       */
13             DEBUG, /** Debugging information meant for developers */
14             INFO, /** Textual information meant for the user */
15             WARNING, /** Warning information concerning the program's execution */
16             ERROR, /** Critical information that requires patch fix */
17             FATAL /** Prioritary level of logging that requires immediate fix */
18         };
19 
20         class Logger
21         {
22         private:
23             std::string _id; /** The logger's unique ID */
24             std::string _file_name; /** The logger's name */
25         public:
26             Logger( const std::string &id, const std::string &file_name );
27             explicit Logger( const std::string &id );
28             ~Logger();
29             std::string get_id();
30             std::string get_file_name();
31             static void log( int level, const std::string &log_message );
32         };
33     }
```

```
34 }
35
36 #endif //SOKOBAN_LOGGER_HPP
```

9.4.2 Logger.hpp

```
1  #include "Logger.hpp"
2
3  #include <iostream>
4
5  #include <boost/locale/generator.hpp>
6
7  #include <boost/date_time/posix_time/posix_time_types.hpp>
8
9  #include <boost/log/core.hpp>
10 #include <boost/log/trivial.hpp>
11 #include <boost/log/expressions.hpp>
12 #include <boost/log/sinks/text_file_backend.hpp>
13 #include <boost/log/utility/setup/file.hpp>
14 #include <boost/log/utility/setup/common_attributes.hpp>
15 #include <boost/log/sources/severity_logger.hpp>
16 #include <boost/log/sources/record_ostream.hpp>
17 #include <boost/log/sources/logger.hpp>
18 #include <boost/log/support/date_time.hpp>
19
20 using namespace sokoban::util;
21 namespace logging = boost::log;
22
23 /**
24  * Injection operator definition meant to show the severity level in string format
25  *
26  * @tparam CharT template's parameter for the character
27  * @tparam TraitsT template's paramenter for the traits
28  * @param strm The output stream
29  * @param lvl The severity level
30  * @return The string containing the severity level
31  */
32 template< typename CharT, typename TraitsT >
33     inline std::basic_ostream< CharT, TraitsT > &operator<<(
34             std::basic_ostream< CharT, TraitsT > &strm, logging::trivial::severity_level lvl )
35     {
36         static const char *const str[] =
37                 {
38                         "trace"
39                         , "debug"
40                         , "info"
41                         , "warning"
42                         , "error"
43                         , "fatal"
44                 };
45         if ( static_cast< std::size_t >(lvl) < ( sizeof( str ) / sizeof( *str ) ) )
46         {
47             strm << str[ lvl ];
48         }
49         else
50         {
51             strm << static_cast< int >(lvl);
52         }
53         return strm;
54     }
55
56 BOOST_LOG_ATTRIBUTE_KEYWORD( severity, "Severity", logging::trivial::severity_level )
57 BOOST_LOG_ATTRIBUTE_KEYWORD( timestamp, "TimeStamp", boost::posix_time::ptime )
58
59 /**
60  * Logger's constructor with a specific filename
61  * @param id The logger's unique I
62  * @param file_name The logger's custom filename
63  */
64 Logger::Logger( const std::string &id, const std::string &file_name )
65 {
66     this->_id = id;
67     this->_file_name = file_name;
68
69     boost::shared_ptr< logging::sinks::synchronous_sink< logging::sinks::text_file_backend > > sink =
        logging::add_file_log
70             (
71                     logging::keywords::file_name = get_file_name(),
72                     logging::keywords::rotation_size = 10 * 1024 * 1024,
73                     logging::keywords::time_based_rotation = logging::sinks::file::rotation_at_time_point(
        0, 0, 0 ),
74                     logging::keywords::format = logging::expressions::stream
75                             << logging::expressions::format_date_time( timestamp, "%Y-%m-%d, %H:%M:%S.%f" )
```

```
76                              << " <" << severity.or_default( logging::trivial::trace )
77                              << "> " << logging::expressions::message,
78                      logging::keywords::auto_flush = true,
79                      logging::keywords::open_mode = std::ios_base::app
80              );
81
82      std::locale loc = boost::locale::generator()( "en_US.UTF-8" );
83      sink->imbue( loc );
84
85      //logging::core::get()->set_filter( severity >= logging::trivial::info );
86
87      logging::add_common_attributes();
88
89      //logging::sources::severity_logger<logging::trivial::severity_level> lg;
90  }
91
92  /**
93   * Logger's constructor initializing its unique id and default name
94   * @param id The unique ID of the logger
95   */
96  Logger::Logger( const std::string &id )
97          : Logger( id, "sokoban.log" )
98  {
99
100 }
101
102 /**
103  * Upon destruction of the logger, print out its identifier
104  */
105 Logger::~Logger()
106 {
107     std::cout << "logger ID: " << get_id() << std::endl;
108 }
109
110 /**
111  * Getter for the logger's unique identifier.
112  * @return The ID
113  */
114 std::string Logger::get_id()
115 {
116     return _id;
117 }
118
119 /**
120  * Getter for the logger's file name
121  * @return The name of the logger.
122  */
123 std::string Logger::get_file_name()
124 {
125     return _file_name;
126 }
127
128 /**
129  * Function that will append the information requested throughout the execution of the program.
130  * @param level The severity level
131  * @param log_message The message to append
132  */
133 void Logger::log( int level, const std::string &log_message )
134 {
135     using namespace logging::trivial;
136
137     const char *message = log_message.c_str();
138
139     switch ( level )
140     {
141         case LoggerLevel::TRACE:
142             BOOST_LOG_TRIVIAL( trace ) << message;
143             break;
144         case LoggerLevel::DEBUG:
145             BOOST_LOG_TRIVIAL( debug ) << message;
146             break;
147         case LoggerLevel::INFO:
148             BOOST_LOG_TRIVIAL( info ) << message;
149             break;
150         case LoggerLevel::WARNING:
151             BOOST_LOG_TRIVIAL( warning ) << message;
152             break;
153         case LoggerLevel::ERROR:
154             BOOST_LOG_TRIVIAL( error ) << message;
155             break;
156         case LoggerLevel::FATAL:
157             BOOST_LOG_TRIVIAL( fatal ) << message;
158             break;
159         default:
```

```
160             BOOST_LOG_TRIVIAL( trace ) << message;
161             break;
162     }
163 }
```

9.5 UI

9.5.1 Category.hpp

```
1  #ifndef SOKOBAN_CATEGORY_HPP
2  #define SOKOBAN_CATEGORY_HPP
3
4  namespace sokoban
5  {
6      namespace ui
7      {
8          namespace Category
9          {
10             /**
11              * Various types of actors and effects could be initialized
12              */
13             enum Type
14             {
15                 None = 0, /** Absolute nothing */
16                 Scene = 1 << 0, /** The actual Scene */
17                 Player = 1 << 1, /** The player itself */
18                 Box = 1 << 2, /** The various boxes */
19                 Platform = 1 << 3, /** The various platforms */
20                 Wall = 1 << 4, /** The various walls */
21                 Player_Movement = 1 << 5, /** The effect concerning the player movement */
22                 Box_Movement = 1 << 6, /** The effect concerning the box's movement */
23                 Sound_Effect = 1 << 7, /** Any various sound effect */
24
25                 Actor = Player | Box | Platform | Wall,
26             };
27         }
28     }
29 }
30
31 #endif //SOKOBAN_CATEGORY_HPP
```

9.5.2 Menu.hpp

```
1  #ifndef SOKOBAN_MENU_HPP
2  #define SOKOBAN_MENU_HPP
3
4  namespace sokoban
5  {
6      namespace ui
7      {
8          class Menu
9          {
10         public:
11             Menu();
12             ~Menu();
13             unsigned short launch_application() const;
14         private:
15             int _res; /** The end result for the game's execution */
16         };
17     }
18 }
19
20 #endif //SOKOBAN_MENU_HPP
```

9.5.3 Menu.cpp

```
1  #include "Menu.hpp"
2
3  #include "gui/Application.hpp"
4
5  using namespace sokoban::ui;
6  using namespace sokoban::ui::gui;
7  using namespace sokoban::util;
8
9  /**
10  * Menu's constructor meant to initialize and execute the game.
11  * Upon execution, a return code is entered.
12  * Should any exception arise, the program's return code will be -1.
13  */
14 Menu::Menu()
15 {
```

```
16      try
17      {
18          Logger::log( LoggerLevel::INFO, "Init Main Frame" );
19          Application main_frame;
20          _res = main_frame.run();
21      }
22      catch ( std::exception &e )
23      {
24          Logger::log( LoggerLevel::ERROR, e.what() );
25          _res = -1;
26      }
27  }
28
29  /**
30   * Menu's destructor
31   */
32  Menu::~Menu()
33  = default;
34
35  /**
36   * Function meant to return the games execution code
37   * @return 0 if run correctly, -1 if not
38   */
39  unsigned short Menu::launch_application() const
40  {
41      return _res;
42  }
```

9.5.4 Resource_Holder.hpp

```
1   #ifndef SOKOBAN_RESOURCE_HOLDER_HPP
2   #define SOKOBAN_RESOURCE_HOLDER_HPP
3
4   #include <map>
5   #include <memory>
6   #include <string>
7   #include <cassert>
8   #include <stdexcept>
9
10  namespace sokoban
11  {
12      namespace ui
13      {
14          /**
15           * Resource holder meant to manage various natures of a Resource.
16           * @tparam Resource Could be a Sprite or a Sound
17           * @tparam Identifier Unique identifier in an enum
18           */
19          template< typename Resource, typename Identifier >
20              class Resource_Holder
21              {
22              public:
23                  /** Loads the requested asset based on its ID and filename */
24                  void load( Identifier id, const std::string &filename );
25                  template< typename Parameter >
26                      /** Loads the requested asset based on its ID and filename plus force a parameter */
27                      void load( Identifier id, const std::string &filename, const Parameter &second_param );
28                  /** Resource getter meant to ensure the retrieval of a non-const Resource */
29                  Resource &get( Identifier id );
30                  /** Resource getter meant to ensure the retrieval of a const Resource */
31                  const Resource &get( Identifier id ) const;
32              private:
33                  /** Map containing the various resources */
34                  std::map< Identifier, std::unique_ptr< Resource > > _resource_map;
35                  /** Insertion of resource into map upon execution of load */
36                  void insert_resource( Identifier id, std::unique_ptr< Resource > resource );
37              };
38
39  #include "Resource_Holder.inl"
40
41      }
42  }
43
44  #endif //SOKOBAN_RESOURCE_HOLDER_HPP
```

9.5.5 Resource_Holder.inl

```
1
2   /**
3    * Function meant to load a resource inside the map containing the various assets
4    * @tparam Resource The type of resource
5    * @tparam Identifier Its unique identifier in the enum
```

```
 * @param id Its unique identifier in the enum
 * @param filename The filename of the asset
 */
template< typename Resource, typename Identifier >
    void Resource_Holder< Resource, Identifier >::load( Identifier id, const std::string &filename )
    {
        // Create and load resource
        std::unique_ptr< Resource > resource( new Resource() );
        if ( !resource->loadFromFile( filename ) )
        {
            throw std::runtime_error( "Resource_Holder::load - Failed to load " + filename );
        }

        // If loading successful, insert resource to map
        insert_resource( id, std::move( resource ) );
    }

/**
 * Function meant to load an asset with a specific parameter that characterizes it.
 * @tparam Resource The type of resource to load
 * @tparam Identifier Its unique identifier within the enum
 * @tparam Parameter The nature of the parameter
 * @param id Its unique identifier within the enum
 * @param filename The filename of the asset
 * @param second_param The parameter that characterizes the asset
 */
template< typename Resource, typename Identifier >
template< typename Parameter >
    void Resource_Holder< Resource, Identifier >::load( Identifier id, const std::string &filename, const
     Parameter &second_param )
    {
        // Create and load resource
        std::unique_ptr< Resource > resource( new Resource() );
        if ( !resource->loadFromFile( filename, second_param ) )
        {
            throw std::runtime_error( "Resource_Holder::load - Failed to load " + filename );
        }

        // If loading successful, insert resource to map
        insert_resource( id, std::move( resource ) );
    }

/**
 * Getter for the asset based on its unique identifier within an enum from a non-const Resource
 * @tparam Resource The type of resource to retrieve
 * @tparam Identifier Its unique identifier within the enum
 * @param id Its unique identifier within the enum
 * @return The asset requested
 */
template< typename Resource, typename Identifier >
    Resource &Resource_Holder< Resource, Identifier >::get( Identifier id )
    {
        auto found = _resource_map.find( id );
        assert( found != _resource_map.end() );

        return *found->second;
    }

/**
 * Getter for the asset based on its unique identifier within an enum from a const Resource
 * @tparam Resource The type of resource to retrieve
 * @tparam Identifier Its unique identifier within the enum
 * @param id Its unique identifier within the enum
 * @return The asset requested
 */
template< typename Resource, typename Identifier >
    const Resource &Resource_Holder< Resource, Identifier >::get( Identifier id ) const
    {
        auto found = _resource_map.find( id );
        assert( found != _resource_map.end() );

        return *found->second;
    }

/**
 * Function meant to insert a new resource inside the Resource Holder's map
 * @tparam Resource The nature of the Resource
 * @tparam Identifier Its unique identifier within the enum
 * @param id Its unique identifier within the enum
 * @param resource The Resource we want to insert
 */
template< typename Resource, typename Identifier >
    void Resource_Holder< Resource, Identifier >::insert_resource( Identifier id, std::unique_ptr< Resource
     > resource )
```

```
88      {
89          // Insert and check success
90          auto inserted = _resource_map.insert( std::make_pair( id, std::move( resource ) ) );
91          assert( inserted.second );
92      }
```

9.5.6 Utility.inl

```
1  #ifndef SOKOBAN_UTILITY_INL
2  #define SOKOBAN_UTILITY_INL
3
4  /**
5   * Function meant to return a stringstream to stream.
6   * @tparam T The nature of the entity that will call this function
7   * @param value The value to insert into the stringstream
8   * @return The stringstream in string format
9   */
10 template< typename T >
11     std::string to_string( const T &value )
12     {
13         std::stringstream stream;
14         stream << value;
15         return stream.str();
16     }
17
18 #endif //SOKOBAN_UTILITY_INL
```

9.6 GUI

9.6.1 Resource_Identifiers.hpp

```
1  #ifndef SOKOBAN_RESOURCE_IDENTIFIERS_HPP
2  #define SOKOBAN_RESOURCE_IDENTIFIERS_HPP
3
4  namespace sf
5  {
6      class Texture;
7      class Font;
8      class Shader;
9      class SoundBuffer;
10 }
11
12 namespace sokoban
13 {
14     namespace ui
15     {
16         namespace gui
17         {
18             namespace Textures
19             {
20                 /**
21                  * The various textures for each actor and entity
22                  */
23                 enum ID
24                 {
25                     Player, /** The player texture */
26                     Box, /** The box texture */
27                     Platform, /** The platform texture */
28                     Wall, /** The box texture */
29                     Background, /** The background texture */
30                     Title_Screen, /** The title screen */
31                     Button, /** The button texture */
32                 };
33             }
34
35             /**
36              * The various shaders that can be displayed on the game
37              */
38             namespace Shaders
39             {
40                 enum ID
41                 {
42                     Brightness_Pass, /** The Brightness levels */
43                     Down_Sample_Pass, /** The samples downscaling */
44                     Gaussian_Blur_Pass, /** The Blur effect */
45                     /* Add_Pass, */
46                 };
47             }
48
49             /**
50              * The various fonts displayed throughout the game
51              */
```

```
52              namespace Fonts
53              {
54                  enum ID
55                  {
56                      Connection_II, /** Minecraft-like font */
57                      Kodomo_Rounded, /** Kiddie Japanese font */
58                      Free_Font, /** GNU Free font */
59                      Rampart_One, /** Bubbly font */
60                  };
61              }
62
63              /**
64               * The various sound effects that can be heard throughout the usage of the software
65               */
66              namespace Sound_Effect
67              {
68                  enum ID
69                  {
70                      Player_Footsteps_Outdoor_Boots,
71                      Player_Footsteps_Outdoor_Boots_On_Wood,
72                      Player_Footsteps_Shoes_Fast_01,
73                      Player_Footsteps_Shoes_On_Wood_01,
74                      Player_Footsteps_Shoes_On_Wood_02,
75                      Player_Footsteps_Snow_01,
76                      Player_Footsteps_Snow_02,
77                      Player_Footsteps_Soft_Fast,
78                      Player_Footsteps_Stair_Ascent_Creaky,
79                      Player_Footsteps_Water_01,
80                      Player_Footsteps_Water_02,
81                      Box_Movement,
82                      Level_Complete,
83                      Game_Complete,
84                      Button_Beep_01,
85                      Button_Beep_02,
86                      Button_Click_01,
87                      Button_Click_02,
88                      Button_Click_03,
89                      Button_Pop_01,
90                      Button_Pop_02,
91                      Button_Switch_01,
92                      Button_Switch_02,
93                  };
94              }
95
96              /**
97               * The various songs that can be played throughout the game
98               */
99              namespace Music
100             {
101                 enum ID
102                 {
103                     Field_Desolate_Domain,
104                     Field_Golden_Harvest,
105                     Field_Greatest_Nation,
106                     Field_Hades_Holiday,
107                     Field_Homeland_Return,
108                     Field_Shattered_Land,
109                     Theme_Courting_The_Princess,
110                     Theme_Festival_Fun,
111                     Theme_Funeral_March,
112                     Theme_Tournament_Time,
113                     Town_Childhood_Home,
114                     Town_Fancy_Castle,
115                     Town_Little_Village,
116                     Town_Nice_Day_Out,
117                     Town_Old_Palace,
118                     Town_Peaceful_Place,
119                     Town_Pleasant_peasants,
120                     Town_Shop_Hopping,
121                     Town_Spooky_Manor,
122                     Town_Tavern_Tune,
123                 };
124             }
125         }
126         template< typename Resource, typename Identifier >
127             class Resource_Holder;
128
129         typedef Resource_Holder< sf::Texture, gui::Textures::ID > Texture_Holder; /** The texture holder */
130         typedef Resource_Holder< sf::Font, gui::Fonts::ID > Font_Holder; /** The font holder */
131         typedef Resource_Holder< sf::Shader, gui::Shaders::ID > Shader_Holder; /** The shader holder */
132         typedef Resource_Holder< sf::SoundBuffer, gui::Sound_Effect::ID > Sound_Buffer_Holder; /** The sound
        effects holder */
133     }
134 }
```

```
135
136 #endif //SOKOBAN_RESOURCE_IDENTIFIERS_HPP
```

9.6.2 Animation.hpp

```
1  #ifndef SOKOBAN_ANIMATION_HPP
2  #define SOKOBAN_ANIMATION_HPP
3
4  #include <SFML/Graphics/Sprite.hpp>
5  #include <SFML/System/Time.hpp>
6
7  namespace sokoban
8  {
9      namespace ui
10     {
11         namespace gui
12         {
13             class Animation
14             : public sf::Drawable
15             , public sf::Transformable
16                 {
17                 public:
18                     Animation();
19                     explicit Animation( const sf::Texture &texture );
20
21                     void set_texture( const sf::Texture &texture );
22                     const sf::Texture *get_texture() const;
23
24                     void set_frame_size( sf::Vector2i frame_size );
25                     sf::Vector2i get_frame_size() const;
26
27                     void set_num_frames( std::size_t num_frames );
28                     std::size_t get_num_frames() const;
29
30                     void set_repeating( bool flag );
31                     bool is_repeating() const;
32
33                     void restart();
34                     bool is_finished() const;
35
36                     sf::FloatRect get_local_bounds() const;
37                     sf::FloatRect get_global_bounds() const;
38
39                     void update( sf::Time dt );
40
41                 private:
42                     sf::Sprite _sprite; /** The current sprite to display */
43                     sf::Vector2i _frame_size; /** The sprite's frame size */
44                     std::size_t _num_frames; /** The number of frames */
45                     std::size_t _current_frame; /** The current frame */
46                     sf::Time _duration; /** The duration of the animation */
47                     sf::Time _elapsed_time; /** The time that went by */
48                     bool _repeat; /** Whether the animation has to be repeated or not */
49                     void draw( sf::RenderTarget &target, sf::RenderStates states ) const override;
50                 };
51         }
52     }
53 }
54
55 #endif //SOKOBAN_ANIMATION_HPP
```

9.6.3 Animation.cpp

```
1  #include "Animation.hpp"
2
3  #include <SFML/Graphics/RenderTarget.hpp>
4  #include <SFML/Graphics/Texture.hpp>
5
6  using namespace sokoban::ui::gui;
7
8  /**
9   * Default constructor for the animation
10  */
11 Animation::Animation()
12 : _sprite()
13 , _frame_size()
14 , _num_frames( 0 )
15 , _current_frame( 0 )
16 , _duration( sf::Time::Zero )
17 , _elapsed_time( sf::Time::Zero )
18 , _repeat( false )
19 {
```

```
20 }
21
22 /**
23  * Default constructor for the animation based on a specific texture
24  * @param texture The texture to initialize
25  */
26 Animation::Animation( const sf::Texture &texture )
27 : _sprite( texture )
28 , _frame_size()
29 , _num_frames( 0 )
30 , _current_frame( 0 )
31 , _duration( sf::Time::Zero )
32 , _elapsed_time( sf::Time::Zero )
33 , _repeat( false )
34 {
35 }
36
37 /**
38  * Setter for the animation's texture
39  * @param texture The new texture to set
40  */
41 void Animation::set_texture( const sf::Texture &texture )
42 {
43     _sprite.setTexture( texture );
44 }
45
46 /**
47  * Getter for the current active texture
48  * @return The texture that is displayed
49  */
50 const sf::Texture *Animation::get_texture() const
51 {
52     return _sprite.getTexture();
53 }
54
55 /**
56  * Setter for the current frame size
57  * @param frame_size The new frame size
58  */
59 void Animation::set_frame_size( sf::Vector2i frame_size )
60 {
61     _frame_size = frame_size;
62 }
63
64 /**
65  * Getter for the frame size
66  * @return The current frame size
67  */
68 sf::Vector2i Animation::get_frame_size() const
69 {
70     return _frame_size;
71 }
72
73 /**
74  * Setter for the number of frames
75  * @param num_frames The new number of frames
76  */
77 void Animation::set_num_frames( std::size_t num_frames )
78 {
79     _num_frames = num_frames;
80 }
81
82 /**
83  * Getter for the number of frames
84  * @return The number of frames
85  */
86 std::size_t Animation::get_num_frames() const
87 {
88     return _num_frames;
89 }
90
91 /**
92  * Setter that defines whether the animation loops or not
93  * @param flag Whether the animation loops or not
94  */
95 void Animation::set_repeating( bool flag )
96 {
97     _repeat = flag;
98 }
99
100 /**
101  * Getter that retrieves whether the animation loops or not
102  * @return whether the animation loops or not
103  */
```

```cpp
bool Animation::is_repeating() const
{
    return _repeat;
}

/**
 * Make the animation restart from the very first frame
 */
void Animation::restart()
{
    _current_frame = 0;
}

/**
 * Function that checks whether an animation has ended or not
 * @return Whether an animation has ended or not
 */
bool Animation::is_finished() const
{
    return _current_frame >= _num_frames;
}

/**
 * Getter for the maximum bounds of an animation
 * @return The maximum bounds of an animation
 */
sf::FloatRect Animation::get_local_bounds() const
{
    return sf::FloatRect( getOrigin(), static_cast< sf::Vector2f >( get_frame_size() ) );
}

/**
 * The globally accessible bounds
 * @return The globally accessible bounds
 */
sf::FloatRect Animation::get_global_bounds() const
{
    return getTransform().transformRect( get_local_bounds() );
}

/**
 * Realtime update each frame of the animation
 * @param dt The clock time
 */
void Animation::update( sf::Time dt )
{
    sf::Time time_per_frame = _duration / static_cast< float >( _num_frames );
    _elapsed_time += dt;

    sf::Vector2i texture_bounds( _sprite.getTexture()->getSize() );
    sf::IntRect texture_rect = _sprite.getTextureRect();

    if( _current_frame == 0 )
    {
        texture_rect = sf::IntRect( 0, 0, _frame_size.x, _frame_size.y );
    }
    while( _elapsed_time >= time_per_frame && ( _current_frame <= _num_frames || _repeat ) )
    {
        texture_rect.left += texture_rect.width;

        if( texture_rect.left + texture_rect.width > texture_bounds.x )
        {
            texture_rect.left = 0;
            texture_rect.top += texture_rect.height;
        }

        _elapsed_time += time_per_frame;

        if( _repeat )
        {
            _current_frame = ( _current_frame + 1 ) % _num_frames;
            if( _current_frame == 0 )
            {
                texture_rect = sf::IntRect( 0, 0, _frame_size.x, _frame_size.y );
            }
            else
            {
                _current_frame++;
            }
        }
    }
    _sprite.setTextureRect( texture_rect );
}

```

```
188 /**
189  * Visually display the various frames of an animation
190  */
191 void Animation::draw( sf::RenderTarget &target, sf::RenderStates states ) const
192 {
193     states.transform *= getTransform();
194     target.draw( _sprite, states );
195 }
```

9.6.4 Application.hpp

```
1  #ifndef SOKOBAN_APPLICATION_HPP
2  #define SOKOBAN_APPLICATION_HPP
3
4  #include "../../util/Logger.hpp"
5  #include "../Resource_Holder.hpp"
6  #include "Resource_Identifiers.hpp"
7  #include "entities/Entity_Player.hpp"
8  #include "states/State_Stack.hpp"
9  #include "Music_Player.hpp"
10 #include "Sound_Player.hpp"
11
12 #include <SFML/System/Time.hpp>
13 #include <SFML/Graphics/Text.hpp>
14 #include <SFML/Graphics/RenderWindow.hpp>
15
16 namespace sokoban
17 {
18     namespace ui
19     {
20         namespace gui
21         {
22             using namespace sokoban::util;
23
24             class Application
25                     : private sf::NonCopyable
26             {
27             public:
28                 Application();
29                 ~Application();
30                 unsigned short run();
31             private:
32                 static const sf::Time _time_per_frame; /** The amount of frames per second we wish to
       display */
33                 sf::RenderWindow _window; /** The window where the drawables are shown */
34                 Texture_Holder _textures; /** The default texture holder */
35                 Font_Holder _fonts; /** The default font holder */
36                 Music_Player _music; /** The default music player */
37                 Sound_Player _sounds; /** The default sound player */
38                 State_Stack _state_stack; /** The stack containing the various states */
39                 sf::Text _statistics_text; /** Text with the FPS amount */
40                 sf::Time _statistics_update_time; /** Timer to update the various statistics */
41                 std::size_t _statistics_num_frames; /** Number of frames per second globally */
42                 void process_input();
43                 void update( const sf::Time &delta_time );
44                 void render();
45                 void update_statistics( sf::Time dt );
46                 void register_states();
47             };
48         }
49     }
50 }
51
52 #endif //SOKOBAN_APPLICATION_HPP
```

9.6.5 Application.cpp

```
1  #include "Application.hpp"
2  #include "../../util/Logger.hpp"
3  #include "Utility.hpp"
4  #include "states/State.hpp"
5  #include "states/State_Identifiers.hpp"
6  #include "states/State_Title.hpp"
7  #include "states/State_Game.hpp"
8  #include "states/State_Menu.hpp"
9  #include "states/State_Pause.hpp"
10 #include "states/State_Settings.hpp"
11
12
13 using namespace sokoban::ui::gui;
14 using namespace sokoban::util;
15
```

```cpp
const sf::Time Application::_time_per_frame = sf::seconds( 1.f / 10.f );

namespace
{
    const int WIDTH = 1920;
    const int HEIGHT = WIDTH / 16 * 9;
    const int BITS_PER_PIXEL = 32;
}

/**
 * Default constructor for the application
 */
Application::Application()
        : _window( sf::VideoMode( WIDTH, HEIGHT, BITS_PER_PIXEL ), "Sokoban", sf::Style::Fullscreen )
          , _textures()
          , _fonts()
          , _music()
          , _sounds()
          , _state_stack( State::Context( _window, _textures, _fonts, _music, _sounds ) )
          , _statistics_text()
          , _statistics_update_time()
          , _statistics_num_frames( 0 )
{
    Logger::log( LoggerLevel::DEBUG, "Setting KeyRepeatedEnabled = false" );
    _window.setKeyRepeatEnabled( false );

    Logger::log( LoggerLevel::DEBUG, "Setting VerticalSyncEnabled = true" );
    _window.setVerticalSyncEnabled( true );

    Logger::log( LoggerLevel::DEBUG, "Loading fonts" );
    _fonts.load( Fonts::Kodomo_Rounded, "assets/fonts/KodomoRounded.otf" );
    _fonts.load( Fonts::Connection_II, "assets/fonts/ConnectionIi-2wj8.otf" );
    _fonts.load( Fonts::Free_Font, "assets/fonts/freefont/FreeSansBold.ttf" );
    _fonts.load( Fonts::Rampart_One, "assets/fonts/RampartOne-Regular.ttf" );

    Logger::log( LoggerLevel::DEBUG, "Loading Title Screen Texture" );
    _textures.load( Textures::Title_Screen, "assets/images/Sample_Sokoban.png" );
    _textures.load( Textures::Button, "assets/images/Buttons.png" );

    Logger::log( LoggerLevel::DEBUG, "Initializing stastistics text" );
    _statistics_text.setFont( _fonts.get( Fonts::Connection_II ) );
    _statistics_text.setPosition( WIDTH / 2.5f, 5.f );
    _statistics_text.setCharacterSize( 10u );
    _statistics_text.setFillColor( sf::Color::Yellow );

    Logger::log( LoggerLevel::DEBUG, "Registering States" );
    register_states();

    Logger::log( LoggerLevel::DEBUG, "Setting Title as first State to load" );
    _state_stack.push_state( States::Title );

    Logger::log( LoggerLevel::DEBUG, "Playing Default Song" );
    _music.play( Music::Town_Pleasant_peasants );
}

/**
 * Application's destructor
 */
Application::~Application()
= default;

/**
 * Realtime update FPS counter
 * @param dt The clock time
 */
void Application::update_statistics( sf::Time dt )
{
    _statistics_update_time += dt;
    _statistics_num_frames += 1;
    if ( _statistics_update_time >= sf::seconds( 1.0f ) )
    {
        _statistics_text.setString(
                "FPS: " + std::to_string( _statistics_num_frames )
        );
        _statistics_update_time -= sf::seconds( 1.0f );
        _statistics_num_frames = 0;
    }
}

/**
 * Globalized Event handler per state within stack
 */
void Application::process_input()
{
```

```
100     sf::Event event{};
101 
102     while ( _window.pollEvent( event ) )
103     {
104         _state_stack.handle_event( event );
105         if ( event.type == sf::Event::Closed )
106         {
107             _window.close();
108         }
109     }
110 
111 }
112 
113 /**
114  * Realtime update all states within stack
115  * @param delta_time The clock time
116  */
117 void Application::update( const sf::Time &delta_time )
118 {
119     _state_stack.update( delta_time );
120 }
121 
122 /**
123  * Visually display everything that makes up the game
124  */
125 void Application::render()
126 {
127     _window.clear();
128     _state_stack.draw();
129     _window.setView( _window.getDefaultView() );
130     _window.draw( _statistics_text );
131     _window.display();
132 }
133 
134 /**
135  * Execute the program and return the execution code
136  * @return 0 if run correctly, -1 if not
137  */
138 unsigned short Application::run()
139 {
140     sf::Clock clock;
141     sf::Time time_since_last_update = sf::Time::Zero;
142     while ( _window.isOpen() )
143     {
144         sf::Time dt = clock.restart();
145         time_since_last_update += dt;
146         while ( time_since_last_update > _time_per_frame )
147         {
148             time_since_last_update -= _time_per_frame;
149             process_input();
150             update( _time_per_frame );
151             if( _state_stack.is_empty() )
152             {
153                 _window.close();
154             }
155         }
156         update_statistics( dt );
157         render();
158     }
159     return 0;
160 }
161 
162 /**
163  * Register the various states within the app
164  */
165 void Application::register_states()
166 {
167     Logger::log( LoggerLevel::DEBUG, "Registering Title State" );
168     _state_stack.register_state< State_Title >( States::Title );
169 
170     Logger::log( LoggerLevel::DEBUG, "Registering Menu State" );
171     _state_stack.register_state< State_Menu >( States::Menu );
172 
173     Logger::log( LoggerLevel::DEBUG, "Registering Game State" );
174     _state_stack.register_state< State_Game >( States::Game );
175 
176     Logger::log( LoggerLevel::DEBUG, "Registering Settings State" );
177     _state_stack.register_state< State_Settings >( States::Settings );
178 
179     Logger::log( LoggerLevel::DEBUG, "Registering Pause State" );
180     _state_stack.register_state< State_Pause >( States::Pause );
181 }
```

9.6.6 Music_Player.hpp

```
#ifndef SOKOBAN_MUSIC_PLAYER_HPP
#define SOKOBAN_MUSIC_PLAYER_HPP

#include "../Resource_Holder.hpp"
#include "Resource_Identifiers.hpp"

#include <SFML/System/NonCopyable.hpp>
#include <SFML/Audio/Music.hpp>

#include <map>
#include <string>

namespace sokoban
{
    namespace ui
    {
        namespace gui
        {
            class Music_Player
        : private sf::NonCopyable
            {
            public:
                Music_Player();
                void play( Music::ID song );
                void stop();
                void set_paused( bool paused );
                void set_volume( float volume );
                float get_volume() const;
            private:
                sf::Music _music; /** The music handler */
                std::map< Music::ID, std::string > _filenames; /** The various files that can be played */
                float _volume; /** The music's volume */
            };
        }
    }
}

#endif //SOKOBAN_MUSIC_PLAYER_HPP
```

9.6.7 Music_Player.cpp

```
#include "Music_Player.hpp"

#include "../../util/Logger.hpp"

using namespace sokoban::ui::gui;
using namespace sokoban::util;

/**
 * Default constructor for the music player
 */
Music_Player::Music_Player()
: _music()
, _filenames()
, _volume( 100.f )
{
    _filenames[ Music::Field_Desolate_Domain ] = "assets/music/Field_-_Desolate_Domain.ogg" ;
    _filenames[ Music::Field_Golden_Harvest ] = "assets/music/Field_-_Golden_Harvest.ogg" ;
    _filenames[ Music::Field_Greatest_Nation ] = "assets/music/Field_-_Greatest_Nation.ogg" ;
    _filenames[ Music::Field_Hades_Holiday ] = "assets/music/Field_-_Hades_Holiday.ogg" ;
    _filenames[ Music::Field_Homeland_Return ] = "assets/music/Field_-_Homeland_Return.ogg" ;
    _filenames[ Music::Field_Shattered_Land ] = "assets/music/Field_-_Shattered_Lands.ogg" ;
    _filenames[ Music::Theme_Courting_The_Princess ] = "assets/music/Theme_-_Courting_The_Princess.ogg" ;
    _filenames[ Music::Theme_Festival_Fun ] = "assets/music/Theme_-_Festival_Fun.ogg" ;
    _filenames[ Music::Theme_Funeral_March ] = "assets/music/Theme_-_Funeral_March.ogg" ;
    _filenames[ Music::Theme_Tournament_Time ] = "assets/music/Theme_-_Tournament_Time.ogg" ;
    _filenames[ Music::Town_Childhood_Home ] = "assets/music/Town_-_Childhood_Home.ogg" ;
    _filenames[ Music::Town_Fancy_Castle ] = "assets/music/Town_-_Fancy_Castle.ogg" ;
    _filenames[ Music::Town_Little_Village ] = "assets/music/Town_-_Little_Village.ogg" ;
    _filenames[ Music::Town_Nice_Day_Out ] = "assets/music/Town_-_Nice_Day_Out.ogg" ;
    _filenames[ Music::Town_Old_Palace ] = "assets/music/Town_-_Old_Palace.ogg" ;
    _filenames[ Music::Town_Peaceful_Place ] = "assets/music/Town_-_Peaceful_Place.ogg" ;
    _filenames[ Music::Town_Pleasant_peasants ] = "assets/music/Town_-_Pleasant_Peasants.ogg" ;
    _filenames[ Music::Town_Shop_Hopping ] = "assets/music/Town_-_Shop_Hopping.ogg" ;
    _filenames[ Music::Town_Spooky_Manor ] = "assets/music/Town_-_Spooky_Manor.ogg" ;
    _filenames[ Music::Town_Tavern_Tune ] = "assets/music/Town_-_Tavern_Tune.ogg" ;
}

/**
 * Function that plays a song based on its ID
```

```
 * @param song The song we wish to play
 */
void Music_Player::play( Music::ID song )
{
    std::string filename = _filenames[ song ];
    if( !_music.openFromFile( filename ) )
    {
        Logger::log( LoggerLevel::ERROR, "Music " + filename + " could not be loaded" );
        throw std::runtime_error( "Music " + filename + " could not be loaded" );
    }
    _music.setVolume( _volume );
    _music.setLoop( true );
    _music.play();
}

/**
 * Function meant to stop a song
 */
void Music_Player::stop()
{
    _music.stop();
}

/**
 * Function meant to pause a song
 * @param paused Whether the song is paused or not
 */
void Music_Player::set_paused( bool paused )
{
    if( paused )
    {
        _music.pause();
    }
    else
    {
        _music.play();
    }
}

/**
 * Volume setter for the song
 * @param volume The value of the volume
 */
void Music_Player::set_volume( float volume )
{
    _volume = volume;
    _music.setVolume( _volume );
}

/**
 * Getter for the volume
 * @return The current value of the volume
 */
float Music_Player::get_volume() const
{
    return _volume;
}
```

9.6.8 Scene_Node.hpp

```
#ifndef SOKOBAN_SCENE_NODE_HPP
#define SOKOBAN_SCENE_NODE_HPP

#include "../Category.hpp"

#include <SFML/System/Time.hpp>
#include <SFML/System/NonCopyable.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include <SFML/Graphics/Transformable.hpp>

#include <set>
#include <memory>
#include <vector>
#include <utility>

namespace sokoban
{
    namespace ui
    {
        namespace gui
        {
            struct Command;
            class Command_Queue;
```

```
24
25              class Scene_Node
26                      : public sf::Transformable
27                        , public sf::Drawable
28                        , private sf::NonCopyable
29              {
30              public:
31                  typedef std::unique_ptr< Scene_Node > Ptr;
32                  typedef std::pair< Scene_Node *, Scene_Node * > Pair;
33                  explicit Scene_Node( Category::Type category = Category::None );
34                  void attach_child( Ptr child );
35                  Ptr detach_child( const Scene_Node &node );
36                  void update( sf::Time dt, Command_Queue &commands );
37                  sf::Vector2f get_world_positions() const;
38                  sf::Transform get_world_transform() const;
39                  void on_command( const Command &command, sf::Time dt );
40                  virtual unsigned int get_category() const;
41                  virtual sf::FloatRect get_bounding_rect() const;
42                  virtual bool is_marked_for_removal() const;
43                  virtual bool is_destroyed() const;
44              private:
45                  std::vector< Ptr > _children; /** The various children that compose a Scene node */
46                  Scene_Node *_parent; /** The parent node */
47                  Category::Type _default_category; /** The default category of each node */
48                  virtual void update_current( sf::Time dt, Command_Queue &commands );
49                  void update_children( sf::Time dt, Command_Queue &commands );
50                  void draw( sf::RenderTarget &target, sf::RenderStates states ) const override;
51                  virtual void draw_current( sf::RenderTarget &target, sf::RenderStates states ) const;
52                  void draw_children( sf::RenderTarget &target, sf::RenderStates states ) const;
53                  void draw_bounding_rect( sf::RenderTarget &target, sf::RenderStates states ) const;
54              };
55          }
56      }
57  }
58
59  #endif //SOKOBAN_SCENE_NODE_HPP
```

9.6.9 Scene_Node.cpp

```
1   #include "Scene_Node.hpp"
2
3   #include "Utility.hpp"
4
5   #include <SFML/Graphics/RectangleShape.hpp>
6   #include <SFML/Graphics/RenderTarget.hpp>
7
8   #include <cmath>
9   #include <cassert>
10  #include <algorithm>
11
12  using namespace sokoban::ui::gui;
13
14  /**
15   * Default constructor for the scene node
16   * @param category The category that characterize the Scene node
17   */
18  Scene_Node::Scene_Node( Category::Type category )
19          : _children()
20            , _parent( nullptr )
21            , _default_category( category )
22  {
23  }
24
25  /**
26   * Function to append a new node child within the Scene node
27   * @param child The new child to append
28   */
29  void Scene_Node::attach_child( Scene_Node::Ptr child )
30  {
31      child->_parent = this;
32      _children.push_back( std::move( child ) );
33  }
34
35  /**
36   * The node that needs to be detached from the children
37   * @param node The node to detach
38   * @return The node that has been detached
39   */
40  Scene_Node::Ptr Scene_Node::detach_child( const Scene_Node &node )
41  {
42      auto found = std::find_if( _children.begin(), _children.end(), [ & ]( Ptr &p )
43      {
44          return p.get() == &node;
```

```
45      } );
46      assert( found != _children.end() );
47
48      Ptr result = std::move( *found );
49      result->_parent = nullptr;
50      _children.erase( found );
51      return result;
52  }
53
54  /**
55   * Function that
56   * @param dt
57   * @param commands
58   */
59  void Scene_Node::update( sf::Time dt, Command_Queue &commands )
60  {
61      update_current( dt, commands );
62      update_children( dt, commands );
63  }
64
65  sf::Vector2f Scene_Node::get_world_positions() const
66  {
67      return get_world_transform() * sf::Vector2f();
68  }
69
70  sf::Transform Scene_Node::get_world_transform() const
71  {
72      sf::Transform transform = sf::Transform::Identity;
73
74      for ( const Scene_Node *node = this; node != nullptr; node = node->_parent )
75      {
76          transform = node->getTransform() * transform;
77      }
78
79      return transform;
80  }
81
82  unsigned int Scene_Node::get_category() const
83  {
84      return _default_category;
85  }
86
87  void Scene_Node::update_current( sf::Time dt, Command_Queue &commands )
88  {
89  }
90
91  void Scene_Node::update_children( sf::Time dt, Command_Queue &commands )
92  {
93      for ( Ptr &child : _children )
94      {
95          child->update( dt, commands );
96      }
97  }
98
99  void Scene_Node::draw( sf::RenderTarget &target, sf::RenderStates states ) const
100 {
101     states.transform *= getTransform();
102     draw_current( target, states );
103     draw_children( target, states );
104     /* FOR DEBUG REASONS */
105     draw_bounding_rect( target, states );
106 }
107
108 void Scene_Node::draw_current( sf::RenderTarget &target, sf::RenderStates states ) const
109 {
110 }
111
112 void Scene_Node::draw_children( sf::RenderTarget &target, sf::RenderStates states ) const
113 {
114     for ( const Ptr &child : _children )
115     {
116         child->draw( target, states );
117     }
118 }
119
120 sf::FloatRect Scene_Node::get_bounding_rect() const
121 {
122     return {};
123 }
124
125 bool Scene_Node::is_marked_for_removal() const
126 {
127     return is_destroyed();
128 }
```

```

bool Scene_Node::is_destroyed() const
{
    return false;
}

void Scene_Node::draw_bounding_rect( sf::RenderTarget &target, sf::RenderStates states ) const
{
    sf::FloatRect rect = get_bounding_rect();
    sf::RectangleShape shape;
    shape.setPosition( sf::Vector2f( rect.left, rect.top ) );
    shape.setSize( sf::Vector2f( rect.width, rect.height ) );
    shape.setFillColor( sf::Color::Transparent );
    shape.setOutlineColor( sf::Color::Green );
    shape.setOutlineThickness( 1.f );
    target.draw( shape );
}
```

9.6.10 Sound_Node.hpp

```
#ifndef SOKOBAN_SOUND_NODE_HPP
#define SOKOBAN_SOUND_NODE_HPP

#include "Scene_Node.hpp"
#include "Resource_Identifiers.hpp"

namespace sokoban
{
    namespace ui
    {
        namespace gui
        {
            class Sound_Player;
            class Sound_Node
        : private Scene_Node
            {
            public:
                explicit Sound_Node( Sound_Player &player );
                void play_sound( Sound_Effect::ID sound, sf::Vector2f position );
                unsigned int get_category() const override;
            private:
                Sound_Player &_sounds; /** The sound player for the current sound node */
            };
        }
    }
}

#endif //SOKOBAN_SOUND_NODE_HPP
```

9.6.11 Sound_Node.cpp

```
#include "Sound_Node.hpp"

#include "Sound_Player.hpp"

using namespace sokoban::ui::gui;

/**
 * Default constructor for the sound node
 * @param player The Sound Player that interacts with the node
 */
Sound_Node::Sound_Node( Sound_Player &player )
: Scene_Node()
, _sounds( player )
{
}

/**
 * Function that plays the sound effect based on its ID and position on the map
 * @param sound The sound effect to play
 * @param position The position on the map
 */
void Sound_Node::play_sound( Sound_Effect::ID sound, sf::Vector2f position )
{
    _sounds.play( sound, position );
}

/**
 * Category getter that characterizes the current node
 * @return Its category being a sound effect
 */
unsigned int Sound_Node::get_category() const
```

```
{
    return Category::Sound_Effect;
}
```

9.6.12 Sound_Player.hpp

```
#ifndef SOKOBAN_SOUND_PLAYER_HPP
#define SOKOBAN_SOUND_PLAYER_HPP

#include "../Resource_Holder.hpp"
#include "Resource_Identifiers.hpp"

#include <SFML/System/Vector2.hpp>
#include <SFML/System/NonCopyable.hpp>
#include <SFML/Audio/SoundBuffer.hpp>
#include <SFML/Audio/Sound.hpp>

#include <list>

namespace sokoban
{
    namespace ui
    {
        namespace gui
        {
            class Sound_Player
            : private sf::NonCopyable
                {
                public:
                    Sound_Player();
                    void play( Sound_Effect::ID effect );
                    void play( Sound_Effect::ID effect, sf::Vector2f position );
                    void remove_stopped_sounds();
                    void set_listener_position( sf::Vector2f position );
                    sf::Vector2f get_listener_position() const;
                    void set_volume( float volume );
                    float get_volume() const;
                private:
                    Sound_Buffer_Holder _sound_buffers; /** Sound Effect Holder */
                    std::list< sf::Sound > _sounds; /** List containing the various sound effects */
                    float _volume; /** The sound effects' volume */
                };
        }
    }
}

#endif //SOKOBAN_SOUND_PLAYER_HPP
```

9.6.13 Sound_Player.cpp

```
#include "Sound_Player.hpp"

#include "../../util/Logger.hpp"

#include <SFML/Audio/Listener.hpp>

#include <cmath>

using namespace sokoban::ui::gui;
using namespace sokoban::util;

namespace
{
    const float Listener_Z = 0.f;
    const float Attenuation = 8.f;
    const float Min_Distance_2D = 0.f;
    const float Min_Distance_3D = std::sqrt( Min_Distance_2D * Min_Distance_2D + Listener_Z * Listener_Z );
}

/**
 * Default constructor for the Sound Player
 */
Sound_Player::Sound_Player()
: _sound_buffers()
, _sounds()
, _volume( 100.f )
{
    _sound_buffers.load( Sound_Effect::Button_Beep_01, "assets/sounds/button_beep_01.ogg" );
    _sound_buffers.load( Sound_Effect::Button_Beep_02, "assets/sounds/button_beep_02.ogg" );
    _sound_buffers.load( Sound_Effect::Button_Click_01, "assets/sounds/button_click_01.ogg" );
    _sound_buffers.load( Sound_Effect::Button_Click_02, "assets/sounds/button_click_02.ogg" );
    _sound_buffers.load( Sound_Effect::Button_Click_03, "assets/sounds/button_click_03.ogg" );
```

```
33      _sound_buffers.load( Sound_Effect::Button_Pop_01, "assets/sounds/button_pop_01.ogg" );
34      _sound_buffers.load( Sound_Effect::Button_Pop_02, "assets/sounds/button_pop_02.ogg" );
35      _sound_buffers.load( Sound_Effect::Player_Footsteps_Outdoor_Boots, "assets/sounds/
        footsteps_outdoor_boots.ogg" );
36      _sound_buffers.load( Sound_Effect::Player_Footsteps_Outdoor_Boots_On_Wood, "assets/sounds/
        footsteps_outdoor_boots_on_wood.ogg" );
37      _sound_buffers.load( Sound_Effect::Player_Footsteps_Shoes_Fast_01, "assets/sounds/
        footsteps_shoes_fast_01.ogg" );
38      _sound_buffers.load( Sound_Effect::Player_Footsteps_Shoes_On_Wood_01, "assets/sounds/
        footsteps_shoes_on_wood_01.ogg" );
39      _sound_buffers.load( Sound_Effect::Player_Footsteps_Shoes_On_Wood_02, "assets/sounds/
        footsteps_shoes_on_wood_02.ogg" );
40      _sound_buffers.load( Sound_Effect::Player_Footsteps_Snow_01, "assets/sounds/footsteps_snow_01.ogg" );
41      _sound_buffers.load( Sound_Effect::Player_Footsteps_Snow_02, "assets/sounds/footsteps_snow_02.ogg" );
42      _sound_buffers.load( Sound_Effect::Player_Footsteps_Soft_Fast, "assets/sounds/footsteps_soft_fast.ogg" )
        ;
43      _sound_buffers.load( Sound_Effect::Player_Footsteps_Stair_Ascent_Creaky, "assets/sounds/
        footsteps_stair_ascent_creaky.ogg" );
44      _sound_buffers.load( Sound_Effect::Player_Footsteps_Water_01, "assets/sounds/footsteps_water_01.ogg" );
45      _sound_buffers.load( Sound_Effect::Player_Footsteps_Water_02, "assets/sounds/footsteps_water_02.ogg" );
46      for( sf::Sound sound : _sounds )
47      {
48          sound.setVolume( _volume );
49      }
50      sf::Listener::setDirection( 0.f, 0.f, -1.f );
51  }
52
53  /**
54   * Function meant to play a sound effect
55   * @param effect The sound effect to play
56   */
57  void Sound_Player::play( Sound_Effect::ID effect )
58  {
59      play( effect, get_listener_position() );
60  }
61
62  /**
63   * Function meant to play a sound effect depending on its position
64   * @param effect The sound effect to play
65   * @param position The position that defines when a sound effect is played
66   */
67  void Sound_Player::play( Sound_Effect::ID effect, sf::Vector2f position )
68  {
69      _sounds.emplace_back( sf::Sound() );
70      sf::Sound &sound = _sounds.back();
71
72      sound.setBuffer( _sound_buffers.get( effect ) );
73      sound.setPosition( position.x, position.y, 0.f );
74      sound.setAttenuation( Attenuation );
75      sound.setMinDistance( Min_Distance_3D );
76      sound.setVolume( _volume );
77
78      sound.play();
79  }
80
81  /**
82   * Function that removes all stopped sounds
83   */
84  void Sound_Player::remove_stopped_sounds()
85  {
86      _sounds.remove_if( [] ( const sf::Sound &s ) {
87          return s.getStatus() == sf::Sound::Stopped;
88      } );
89  }
90
91  /**
92   * Set the sound effect position
93   * @param position The position meant to be listened
94   */
95  void Sound_Player::set_listener_position( sf::Vector2f position )
96  {
97      sf::Listener::setPosition( position.x, position.y, Listener_Z );
98  }
99
100 /**
101  * Getter for the sound effect position
102  * @return The position of the sound effect
103  */
104 sf::Vector2f Sound_Player::get_listener_position() const
105 {
106     sf::Vector3f position = sf::Listener::getPosition();
107     return { position.x, position.y };
108 }
109
```

```
110 /**
111  * Volume setter for the various sound effects
112  * @param volume The value of volume to set
113  */
114 void Sound_Player::set_volume( float volume )
115 {
116     _volume = volume;
117     for( sf::Sound sound : _sounds )
118     {
119         sound.setVolume( _volume );
120     }
121 }
122 
123 /**
124  * Getter for the value of volume
125  * @return The value of the sound effects volume
126  */
127 float Sound_Player::get_volume() const
128 {
129     return _volume;
130 }
```

9.6.14 Sprite_Node.hpp

```
1  #ifndef SOKOBAN_SPRITE_NODE_HPP
2  #define SOKOBAN_SPRITE_NODE_HPP
3  
4  #include "Scene_Node.hpp"
5  
6  #include <SFML/Graphics/Sprite.hpp>
7  
8  namespace sokoban
9  {
10     namespace ui
11     {
12         namespace gui
13         {
14             class Sprite_Node
15                     : public Scene_Node
16             {
17             public:
18                 explicit Sprite_Node( const sf::Texture &texture );
19                 Sprite_Node( const sf::Texture &texture, const sf::IntRect &textureRect );
20                 void set_texture( const sf::Texture &texture );
21                 void set_texture( const sf::Texture &texture, const sf::IntRect &textureRect );
22             private:
23                 sf::Sprite _sprite; /** The sprite to display */
24                 void draw_current( sf::RenderTarget &target, sf::RenderStates states ) const override;
25             };
26         }
27     }
28 }
29 
30 #endif //SOKOBAN_SPRITE_NODE_HPP
```

9.6.15 Sprite_Node.cpp

```
1  #include "Sprite_Node.hpp"
2  
3  #include <SFML/Graphics/RenderTarget.hpp>
4  
5  using namespace sokoban::ui::gui;
6  
7  /**
8   * Default constructor for the sprite node
9   * @param texture The texture to display the sprite with
10  */
11 Sprite_Node::Sprite_Node( const sf::Texture &texture )
12         : _sprite( texture )
13 {
14 }
15 
16 /**
17  * Default constructor for the sprite node
18  * @param texture The texture to display the sprite with
19  * @param textureRect The coordinates of a texture within a sprite sheet
20  */
21 Sprite_Node::Sprite_Node( const sf::Texture &texture, const sf::IntRect &textureRect )
22         : _sprite( texture, textureRect )
23 {
24 
25 }
```

```
26
27 /**
28  * Setter for the sprite's texture
29  * @param texture The new texture to display
30  */
31 void Sprite_Node::set_texture( const sf::Texture &texture )
32 {
33     _sprite.setTexture( texture );
34 }
35
36 /**
37  * Setter for the sprite's texture
38  * @param texture The new texture to display
39  * @param textureRect The coordinates of the texture within a sprite sheet
40  */
41 void Sprite_Node::set_texture( const sf::Texture &texture, const sf::IntRect &textureRect )
42 {
43     set_texture( texture );
44     _sprite.setTextureRect( textureRect );
45 }
46
47 /**
48  * Visually display the sprite node
49  * @param target The target to display the sprite onto
50  * @param states The various states that characterize the window
51  */
52 void Sprite_Node::draw_current( sf::RenderTarget &target, sf::RenderStates states ) const
53 {
54     target.draw( _sprite, states );
55 }
```

9.6.16 Utility.hpp

```
1  #ifndef SOKOBAN_UTILITY_HPP
2  #define SOKOBAN_UTILITY_HPP
3
4  #include <SFML/System/Vector2.hpp>
5  #include <SFML/Window/Keyboard.hpp>
6
7  #include <sstream>
8
9  namespace sf
10 {
11     class Sprite;
12     class Text;
13 }
14
15 namespace sokoban
16 {
17     namespace ui
18     {
19         namespace gui
20         {
21             class Animation;
22             class Utility
23             {
24             public:
25                 template< typename T >
26                     std::string to_string( const T &value );
27                 std::string to_string( sf::Keyboard::Key key );
28                 static void center_origin( sf::Sprite &sprite );
29                 static void center_origin( sf::Text &text );
30                 static void center_origin( Animation &animation );
31                 float to_degree( float radian );
32                 float to_radian( float degree );
33                 int random_int( int exclusive_max );
34                 float length( sf::Vector2f vector );
35                 sf::Vector2f unit_vector( sf::Vector2f vector );
36             };
37 #include "../Utility.inl"
38         }
39     }
40 }
41
42 #endif //SOKOBAN_UTILITY_HPP
```

9.6.17 Utility.cpp

```
1 #include "Utility.hpp"
2 #include "Animation.hpp"
3 #include "components/Button.hpp"
4
```

```
5  #include <SFML/Graphics/Text.hpp>
6  #include <SFML/Graphics/Sprite.hpp>
7
8  #include <random>
9  #include <cmath>
10 #include <ctime>
11 #include <cassert>
12
13 using namespace sokoban::ui::gui;
14
15 namespace
16 {
17     std::default_random_engine create_Random_Engine()
18     {
19         auto seed = static_cast< unsigned long >( std::time( nullptr ) );
20         return std::default_random_engine( seed );
21     }
22     auto Random_Engine = create_Random_Engine();
23 }
24
25 /**
26  * Function meant to return a keyboard key to string
27  * @param key The key to be converted
28  * @return The key in textual form
29  */
30 std::string Utility::to_string( sf::Keyboard::Key key )
31 {
32 #define BOOK_KEY_TO_STRING_CASE( KEY ) case sf::Keyboard::KEY: return #KEY;
33
34     switch ( key )
35     {
36         BOOK_KEY_TO_STRING_CASE(Unknown)
37         BOOK_KEY_TO_STRING_CASE(A)
38         BOOK_KEY_TO_STRING_CASE(B)
39         BOOK_KEY_TO_STRING_CASE(C)
40         BOOK_KEY_TO_STRING_CASE(D)
41         BOOK_KEY_TO_STRING_CASE(E)
42         BOOK_KEY_TO_STRING_CASE(F)
43         BOOK_KEY_TO_STRING_CASE(G)
44         BOOK_KEY_TO_STRING_CASE(H)
45         BOOK_KEY_TO_STRING_CASE(I)
46         BOOK_KEY_TO_STRING_CASE(J)
47         BOOK_KEY_TO_STRING_CASE(K)
48         BOOK_KEY_TO_STRING_CASE(L)
49         BOOK_KEY_TO_STRING_CASE(M)
50         BOOK_KEY_TO_STRING_CASE(N)
51         BOOK_KEY_TO_STRING_CASE(O)
52         BOOK_KEY_TO_STRING_CASE(P)
53         BOOK_KEY_TO_STRING_CASE(Q)
54         BOOK_KEY_TO_STRING_CASE(R)
55         BOOK_KEY_TO_STRING_CASE(S)
56         BOOK_KEY_TO_STRING_CASE(T)
57         BOOK_KEY_TO_STRING_CASE(U)
58         BOOK_KEY_TO_STRING_CASE(V)
59         BOOK_KEY_TO_STRING_CASE(W)
60         BOOK_KEY_TO_STRING_CASE(X)
61         BOOK_KEY_TO_STRING_CASE(Y)
62         BOOK_KEY_TO_STRING_CASE(Z)
63         BOOK_KEY_TO_STRING_CASE(Num0)
64         BOOK_KEY_TO_STRING_CASE(Num1)
65         BOOK_KEY_TO_STRING_CASE(Num2)
66         BOOK_KEY_TO_STRING_CASE(Num3)
67         BOOK_KEY_TO_STRING_CASE(Num4)
68         BOOK_KEY_TO_STRING_CASE(Num5)
69         BOOK_KEY_TO_STRING_CASE(Num6)
70         BOOK_KEY_TO_STRING_CASE(Num7)
71         BOOK_KEY_TO_STRING_CASE(Num8)
72         BOOK_KEY_TO_STRING_CASE(Num9)
73         BOOK_KEY_TO_STRING_CASE(Escape)
74         BOOK_KEY_TO_STRING_CASE(LControl)
75         BOOK_KEY_TO_STRING_CASE(LShift)
76         BOOK_KEY_TO_STRING_CASE(LAlt)
77         BOOK_KEY_TO_STRING_CASE(LSystem)
78         BOOK_KEY_TO_STRING_CASE(RControl)
79         BOOK_KEY_TO_STRING_CASE(RShift)
80         BOOK_KEY_TO_STRING_CASE(RAlt)
81         BOOK_KEY_TO_STRING_CASE(RSystem)
82         BOOK_KEY_TO_STRING_CASE(Menu)
83         BOOK_KEY_TO_STRING_CASE(LBracket)
84         BOOK_KEY_TO_STRING_CASE(RBracket)
85         BOOK_KEY_TO_STRING_CASE(SemiColon)
86         BOOK_KEY_TO_STRING_CASE(Comma)
87         BOOK_KEY_TO_STRING_CASE(Period)
88         BOOK_KEY_TO_STRING_CASE(Quote)
```

```
89          BOOK_KEY_TO_STRING_CASE(Slash)
90          BOOK_KEY_TO_STRING_CASE(BackSlash)
91          BOOK_KEY_TO_STRING_CASE(Tilde)
92          BOOK_KEY_TO_STRING_CASE(Equal)
93          BOOK_KEY_TO_STRING_CASE(Dash)
94          BOOK_KEY_TO_STRING_CASE(Space)
95          BOOK_KEY_TO_STRING_CASE(Return)
96          BOOK_KEY_TO_STRING_CASE(BackSpace)
97          BOOK_KEY_TO_STRING_CASE(Tab)
98          BOOK_KEY_TO_STRING_CASE(PageUp)
99          BOOK_KEY_TO_STRING_CASE(PageDown)
100         BOOK_KEY_TO_STRING_CASE(End)
101         BOOK_KEY_TO_STRING_CASE(Home)
102         BOOK_KEY_TO_STRING_CASE(Insert)
103         BOOK_KEY_TO_STRING_CASE(Delete)
104         BOOK_KEY_TO_STRING_CASE(Add)
105         BOOK_KEY_TO_STRING_CASE(Subtract)
106         BOOK_KEY_TO_STRING_CASE(Multiply)
107         BOOK_KEY_TO_STRING_CASE(Divide)
108         BOOK_KEY_TO_STRING_CASE(Left)
109         BOOK_KEY_TO_STRING_CASE(Right)
110         BOOK_KEY_TO_STRING_CASE(Up)
111         BOOK_KEY_TO_STRING_CASE(Down)
112         BOOK_KEY_TO_STRING_CASE(Numpad0)
113         BOOK_KEY_TO_STRING_CASE(Numpad1)
114         BOOK_KEY_TO_STRING_CASE(Numpad2)
115         BOOK_KEY_TO_STRING_CASE(Numpad3)
116         BOOK_KEY_TO_STRING_CASE(Numpad4)
117         BOOK_KEY_TO_STRING_CASE(Numpad5)
118         BOOK_KEY_TO_STRING_CASE(Numpad6)
119         BOOK_KEY_TO_STRING_CASE(Numpad7)
120         BOOK_KEY_TO_STRING_CASE(Numpad8)
121         BOOK_KEY_TO_STRING_CASE(Numpad9)
122         BOOK_KEY_TO_STRING_CASE(F1)
123         BOOK_KEY_TO_STRING_CASE(F2)
124         BOOK_KEY_TO_STRING_CASE(F3)
125         BOOK_KEY_TO_STRING_CASE(F4)
126         BOOK_KEY_TO_STRING_CASE(F5)
127         BOOK_KEY_TO_STRING_CASE(F6)
128         BOOK_KEY_TO_STRING_CASE(F7)
129         BOOK_KEY_TO_STRING_CASE(F8)
130         BOOK_KEY_TO_STRING_CASE(F9)
131         BOOK_KEY_TO_STRING_CASE(F10)
132         BOOK_KEY_TO_STRING_CASE(F11)
133         BOOK_KEY_TO_STRING_CASE(F12)
134         BOOK_KEY_TO_STRING_CASE(F13)
135         BOOK_KEY_TO_STRING_CASE(F14)
136         BOOK_KEY_TO_STRING_CASE(F15)
137         BOOK_KEY_TO_STRING_CASE(Pause)
138         case sf::Keyboard::KeyCount:
139             break;
140     }
141     return "";
142 }
143
144 /**
145  * Function that sets a sprite's origin in its center
146  * @param sprite The sprite to center
147  */
148 void Utility::center_origin( sf::Sprite &sprite )
149 {
150     sf::FloatRect bounds = sprite.getLocalBounds();
151     sprite.setOrigin(
152             std::floor( bounds.left + bounds.width / 2.f ),
153             std::floor( bounds.top + bounds.height / 2.f )
154             );
155 }
156
157 /**
158  * Function that sets a text's origin in its center
159  * @param text The text to center
160  */
161 void Utility::center_origin( sf::Text &text )
162 {
163     sf::FloatRect bounds = text.getLocalBounds();
164     text.setOrigin(
165             std::floor( bounds.left + bounds.width / 2.f ),
166             std::floor( bounds.top + bounds.height / 2.f )
167             );
168 }
169
170 /**
171  * Function that sets the animation's origin in its center
172  * @param animation The animation to center
```

```
 */
void Utility::center_origin( Animation &animation )
{
    sf::FloatRect bounds = animation.get_local_bounds();
    animation.setOrigin(
            std::floor( bounds.left + bounds.width / 2.f ),
            std::floor( bounds.top + bounds.height / 2.f )
    );
}

/**
 * Function to convert from Radian to Degrees
 * @param radian The radian value to convert
 * @return The radian value in degrees form
 */
float Utility::to_degree( float radian )
{
    return 100.f / M_PI * radian;
}

/**
 * Function to convert from Degree to Radian
 * @param degree The degree value to convert
 * @return The degree value in radian form
 */
float Utility::to_radian( float degree )
{
    return M_PI / 100.f * degree;
}

/**
 * Random integer generator
 * @param exclusive_max The maximum value we wish to obtain
 * @return A value from 0 to Max
 */
int Utility::random_int( int exclusive_max )
{
    std::uniform_int_distribution<> dist( 0, exclusive_max - 1 );
    return dist( Random_Engine );
}

/**
 * Function to calculate the distance between within a vector
 * @param vector The vecto's to calculate
 * @return X * X + Y * Y
 */
float Utility::length( sf::Vector2f vector )
{
    return std::sqrt( vector.x * vector.x + vector.y * vector.y );
}

/**
 * Unit calculator based on the length of a vector and a vector itself
 */
sf::Vector2f Utility::unit_vector( sf::Vector2f vector )
{
    assert( vector != sf::Vector2f( 0.f, 0.f ) );
    return vector / length( vector );
}
```

9.6.18 World.hpp

```
#ifndef SOKOBAN_WORLD_HPP
#define SOKOBAN_WORLD_HPP

#include "../Resource_Holder.hpp"
#include "Resource_Identifiers.hpp"
#include "Scene_Node.hpp"
#include "Sprite_Node.hpp"
#include "entities/Entity.hpp"
#include "entities/Entity_Box.hpp"
#include "entities/Entity_Wall.hpp"
#include "entities/Entity_Player.hpp"
#include "entities/Entity_Platform.hpp"
#include "Sound_Player.hpp"
#include "../../model/Board.hpp"

#include <SFML/System/NonCopyable.hpp>
#include <SFML/Graphics/View.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/Graphics/Text.hpp>

#include <queue>
```

```
22 #include <vector>
23
24 namespace sf
25 {
26     class RenderTarget;
27 }
28
29 namespace sokoban
30 {
31     namespace ui
32     {
33         namespace gui
34         {
35             class World
36         : private sf::NonCopyable
37             {
38             public:
39                 World( sf::RenderTarget &target, const model::Board &board, Font_Holder &fonts, Sound_Player
       &sounds );
40                 ~World();
41                 void update( sf::Time dt );
42                 void draw();
43                 bool is_board_completed() const;
44                 void move_up( bool pressed );
45                 void move_down( bool pressed );
46                 void move_left( bool pressed );
47                 void move_right( bool pressed );
48                 void set_reset_counter( int reset_counter );
49                 int get_reset_counter() const;
50             private:
51                 /**
52                  * The various background colors
53                  */
54                 enum class Background_Color
55                 {
56                     CONCRETE,
57                     DIRT,
58                     GRASS,
59                     SAND
60                 };
61                 sf::RenderTarget &_target; /** The window that will display the various components */
62                 sf::View _world_view; /** The globally defined world view */
63                 Texture_Holder _textures; /** Default texture holder */
64                 Font_Holder &_fonts; /** Default font holder */
65                 Sound_Player &_sounds; /** Default Sound Player */
66                 Scene_Node _scene_graph; /** Main scene node */
67                 std::vector< Scene_Node * > _scene_layers; /** Various Scene layers */
68                 bool _player_is_moving_up; /** Whether the player is moving up */
69                 bool _player_is_moving_down; /** Whether the player is moving down */
70                 bool _player_is_moving_left; /** Whether the player is moving left */
71                 bool _player_is_moving_right; /** Whether the player is moving right */
72                 sf::FloatRect _world_bounds; /** The accessible limits of the window */
73                 model::Board _board; /** The Board containing all the actors */
74                 Sprite_Node *_player_sprite; /** The player sprite */
75                 model::Player *_board_player; /** The player within the board */
76                 entity::Entity_Player *_player_entity; /** The player entity containing the various assets
       coordinates */
77                 std::vector< Sprite_Node * > _box_sprites; /** The various box sprites */
78                 std::vector< model::Box * > _box_actors; /** The various box actors within the board */
79                 std::vector< entity::Entity_Box * > _box_entities; /** The various box entities containing
       the assets' coordinates */
80                 std::vector< entity::Entity * > _entities; /** The entirety of the entities present in the
       world */
81                 sf::Texture *_box_texture_sheet; /** The box texture */
82                 sf::Texture *_platform_texture_sheet; /** The platform texture */
83                 sf::Texture *_wall_texture_sheet; /** The wall texture */
84                 sf::Texture *_player_texture_sheet; /** The player texture */
85                 sf::Texture *_background_texture; /** The background texture */
86                 sf::Text *_text; /** The visually visible text */
87                 int _reset_counter; /** The amount of resets */
88                 void load_textures();
89                 void build_scene();
90             };
91         }
92     }
93 }
94
95 #endif //SOKOBAN_WORLD_HPP
```

9.6.19 World.cpp

```
1 #include "World.hpp"
2
```

```
3  #include "../../util/Logger.hpp"
4  #include "Sound_Node.hpp"
5
6  #include <SFML/Graphics/RenderTarget.hpp>
7  #include <SFML/Graphics/Text.hpp>
8
9  #include <cmath>
10 #include <ctime>
11 #include <random>
12 #include <utility>
13 #include <sstream>
14 #include <iostream>
15
16 using namespace sokoban::ui::gui;
17 using namespace sokoban::util;
18
19 namespace
20 {
21     int steps_counter = 0;
22 }
23
24 /**
25  * Default constructor for the world
26  * @param target The target to display the sprites onto
27  * @param board The board containing the various actors and the level's skeleton
28  * @param fonts The various fonts used
29  * @param sounds The various sound effects
30  */
31 World::World( sf::RenderTarget &target, const model::Board &board, Font_Holder &fonts, Sound_Player &sounds
        )
32 : _target( target )
33   , _world_view( target.getDefaultView() )
34   , _textures()
35   , _fonts( fonts )
36   , _sounds( sounds )
37   , _scene_graph()
38   , _scene_layers()
39   , _world_bounds( 0.f, 0.f, _world_view.getSize().x, _world_view.getSize().y )
40   , _player_is_moving_up( false )
41   , _player_is_moving_down( false )
42   , _player_is_moving_left( false )
43   , _player_is_moving_right( false )
44   , _board_player( nullptr )
45   , _player_entity( nullptr )
46   , _player_sprite( nullptr )
47   , _box_texture_sheet( nullptr )
48   , _platform_texture_sheet( nullptr )
49   , _wall_texture_sheet( nullptr )
50   , _player_texture_sheet( nullptr )
51   , _background_texture( nullptr )
52   , _board( "" )
53   , _text( nullptr )
54   , _reset_counter( 0 )
55 {
56     _board = board;
57     Logger::log( LoggerLevel::DEBUG, "Level Layout:" );
58     for( model::Actor *actor : _board.get_world() )
59     {
60         Logger::log( LoggerLevel::DEBUG, actor->to_string() );
61     }
62     /*step_buffer.loadFromFile( "assets/sounds/footsteps_outdoor_boots.ogg" );
63     step_sound.setBuffer( step_buffer );
64     step_sound.setVolume( 10.f );
65     box_move_buffer.loadFromFile( "assets/sounds/wood_creak_01.ogg" );
66     box_move_sound.setBuffer( box_move_buffer );
67     box_move_sound.setVolume( 10.f );*/
68     load_textures();
69     build_scene();
70     steps_counter = 0;
71 }
72
73 /**
74  * Default destructor for the World
75  */
76 World::~World()
77 {
78     delete _text;
79     for ( Scene_Node *layer: _scene_layers )
80     {
81         delete layer;
82     }
83     for( entity::Entity *entity : _entities )
84     {
85         delete entity;
```

```
86      }
87      _scene_layers.clear();
88      _entities.clear();
89      delete _player_texture_sheet;
90      delete _box_texture_sheet;
91      delete _platform_texture_sheet;
92      delete _wall_texture_sheet;
93      delete _background_texture;
94      _box_sprites.clear();
95      _box_entities.clear();
96  }
97
98  /**
99   * Realtime updates the visually available entities
100  * @param dt The clock time
101  */
102 void World::update( sf::Time dt )
103 {
104     float SPACE = 64.f;
105     sf::IntRect player_assets_coords;
106     float player_x_coords;
107     float player_y_coords;
108     float player_width_coords;
109     float player_height_coords;
110     if ( _player_is_moving_up )
111     {
112         auto player_asset_rect = _player_entity->get_player_face_map().find( entity::Entity_Player::Face::
        NORTH )->second;
113         player_x_coords = player_asset_rect.at( 0 );
114         player_y_coords = player_asset_rect.at( 1 );
115         player_width_coords = player_asset_rect.at( 2 );
116         player_height_coords = player_asset_rect.at( 3 );
117         player_assets_coords = sf::IntRect( player_x_coords, player_y_coords, player_width_coords,
        player_height_coords );
118         _player_sprite->set_texture( *_player_texture_sheet );
119         _player_sprite->set_texture( *_player_texture_sheet, player_assets_coords );
120         if ( _board.check_wall_collision( _board_player, _board.TOP_COLLISION ) )
121         {
122             return;
123         }
124         if ( _board.check_box_collision( _board.TOP_COLLISION ) )
125         {
126             return;
127         }
128         else
129         {
130             for ( int i = 0; i < _box_entities.size(); i++ )
131             {
132                 _box_sprites.at( i )->setPosition( _box_actors.at( i )->get_x(), _box_actors.at( i )->get_y
        () );
133             }
134         }
135         _player_sprite->move( 0.f, -SPACE );
136         _player_entity->set_y( _player_entity->get_y() - SPACE );
137         _board_player->set_y( _board_player->get_y() - SPACE );
138     }
139     if ( _player_is_moving_down )
140     {
141         auto player_asset_rect = _player_entity->get_player_face_map().find( entity::Entity_Player::Face::
        SOUTH )->second;
142         player_x_coords = player_asset_rect.at( 0 );
143         player_y_coords = player_asset_rect.at( 1 );
144         player_width_coords = player_asset_rect.at( 2 );
145         player_height_coords = player_asset_rect.at( 3 );
146         player_assets_coords = sf::IntRect( player_x_coords, player_y_coords, player_width_coords,
        player_height_coords );
147         _player_sprite->set_texture( *_player_texture_sheet, player_assets_coords );
148         if ( _board.check_wall_collision( _board_player, _board.BOTTOM_COLLISION ) )
149         {
150             return;
151         }
152         if ( _board.check_box_collision( _board.BOTTOM_COLLISION ) )
153         {
154             return;
155         }
156         else
157         {
158             for ( int i = 0; i < _box_entities.size(); i++ )
159             {
160                 _box_sprites.at( i )->setPosition( _box_actors.at( i )->get_x(), _box_actors.at( i )->get_y
        () );
161             }
162         }
163         _player_sprite->move( 0.f, +SPACE );
```

```
164         _player_entity->set_y( _player_entity->get_y() + SPACE );
165         _board_player->set_y( _board_player->get_y() + SPACE );
166     }
167     if ( _player_is_moving_left )
168     {
169         auto player_asset_rect = _player_entity->get_player_face_map().find( entity::Entity_Player::Face::
    WEST )->second;
170         player_x_coords = player_asset_rect.at( 0 );
171         player_y_coords = player_asset_rect.at( 1 );
172         player_width_coords = player_asset_rect.at( 2 );
173         player_height_coords = player_asset_rect.at( 3 );
174         player_assets_coords = sf::IntRect( player_x_coords, player_y_coords, player_width_coords,
    player_height_coords );
175         _player_sprite->set_texture( *_player_texture_sheet, player_assets_coords );
176         if ( _board.check_wall_collision( _board_player, _board.LEFT_COLLISION ) )
177         {
178             return;
179         }
180         if ( _board.check_box_collision( _board.LEFT_COLLISION ) )
181         {
182             return;
183         }
184         else
185         {
186             for ( int i = 0; i < _box_entities.size(); i++ )
187             {
188                 _box_sprites.at( i )->setPosition( _box_actors.at( i )->get_x(), _box_actors.at( i )->get_y
    () );
189             }
190         }
191         _player_sprite->move( -SPACE, 0.f );
192         _player_entity->set_x( _player_entity->get_x() - SPACE );
193         _board_player->set_x( _board_player->get_x() - SPACE );
194     }
195     if ( _player_is_moving_right )
196     {
197         auto player_asset_rect = _player_entity->get_player_face_map().find( entity::Entity_Player::Face::
    EAST )->second;
198         player_x_coords = player_asset_rect.at( 0 );
199         player_y_coords = player_asset_rect.at( 1 );
200         player_width_coords = player_asset_rect.at( 2 );
201         player_height_coords = player_asset_rect.at( 3 );
202         player_assets_coords = sf::IntRect( player_x_coords, player_y_coords, player_width_coords,
    player_height_coords );
203         _player_sprite->set_texture( *_player_texture_sheet, player_assets_coords );
204         if ( _board.check_wall_collision( _board_player, _board.RIGHT_COLLISION ) )
205         {
206             return;
207         }
208         if ( _board.check_box_collision( _board.RIGHT_COLLISION ) )
209         {
210             return;
211         }
212         else
213         {
214             for ( int i = 0; i < _box_entities.size(); i++ )
215             {
216                 _box_sprites.at( i )->setPosition( _box_actors.at( i )->get_x(), _box_actors.at( i )->get_y
    () );
217             }
218         }
219         _player_sprite->move( +SPACE, 0.f );
220         _board_player->set_x( _board_player->get_x() + SPACE );
221     }
222     _text->setString(
223             "Steps:  " + std::to_string( steps_counter ) + "\n" +
224             "Resets:  " + std::to_string( get_reset_counter() ) + "\n" +
225             "Move Up: Arrow Up" + "\n" +
226             "Move Down: Arrow Down" + "\n" +
227             "Move Left: Arrow Left" + "\n" +
228             "Move Right: Arrow Right" + "\n" +
229             "Reset Board: R" + "\n" +
230             "Skip Level: S" + "\n" +
231             "Undo Skip: X" + "\n" +
232             "Pause Game: Escape" + "\n"
233             );
234     if ( _board.is_completed() )
235     {
236         //step_sound.stop();
237         /* TODO: BLINKING TEXT WHEN FINISHED */
238     }
239
240 }
241
```

```
242 /**
243  * Visually display the various scene nodes that make up the world
244  */
245 void World::draw()
246 {
247     _target.setView( _world_view );
248     for ( Scene_Node *layer: _scene_layers )
249     {
250         _target.draw( *layer );
251     }
252     _target.draw( *_text );
253 }
254 
255 /**
256  * Load the various sprite sheets
257  */
258 void World::load_textures()
259 {
260     Logger::log( LoggerLevel::INFO, "Loading Textures..." );
261     _player_texture_sheet = new sf::Texture();
262     _player_texture_sheet->loadFromFile( "assets/images/Spritesheet/character_spritesheet.png" );
263     _box_texture_sheet = new sf::Texture();
264     _box_texture_sheet->loadFromFile( "assets/images/Spritesheet/boxes_spritesheet.png" );
265     _platform_texture_sheet = new sf::Texture();
266     _platform_texture_sheet->loadFromFile( "assets/images/Spritesheet/platforms_spritesheet.png" );
267     _wall_texture_sheet = new sf::Texture();
268     _wall_texture_sheet->loadFromFile( "assets/images/Spritesheet/wall_round_spritesheet.png" );
269     _background_texture = new sf::Texture();
270 }
271 
272 /**
273  * Build the scene based on the board's skeleton
274  */
275 void World::build_scene()
276 {
277     Logger::log( LoggerLevel::INFO, "World Node init" );
278     _player_is_moving_up = false;
279     _player_is_moving_down = false;
280     _player_is_moving_left = false;
281     _player_is_moving_right = false;
282     _text = new sf::Text();
283     _text->setFont( _fonts.get( Fonts::Connection_II ) );
284     _text->setPosition(
285             _target.getSize().x - 350.f,
286             20.f );
287     _text->setCharacterSize( 24 );
288     _text->setFillColor( sf::Color::Black );
289 
290     Logger::log( LoggerLevel::INFO, "Board size: " + std::to_string( _board.get_world().size() ) );
291     _box_sprites = std::vector< Sprite_Node * >();
292     _box_actors = std::vector< model::Box * >();
293     _box_entities = std::vector< entity::Entity_Box * >();
294 
295     Logger::log( LoggerLevel::INFO, "Building Scene..." );
296     std::size_t world_size = _board.get_world().size() + 1;
297     _scene_layers = std::vector< Scene_Node * >();
298     std::stringstream ss;
299     ss << "Initializing " << std::to_string( world_size ) << " Scene nodes...";
300     Logger::log( LoggerLevel::DEBUG, ss.str() );
301 
302     std::random_device rd;
303     std::mt19937 mt( rd() );
304 
305     int min_within_enum;
306     int max_within_enum;
307 
308     min_within_enum = static_cast< int >( Background_Color::CONCRETE );
309     max_within_enum = static_cast< int >( Background_Color::SAND );
310     std::uniform_int_distribution< int > background_distribution( min_within_enum, max_within_enum );
311     auto random_background_color = static_cast< Background_Color >( background_distribution( mt ) );
312     switch ( random_background_color )
313     {
314         case Background_Color::CONCRETE:
315             _background_texture->loadFromFile( "assets/images/PNG/GroundGravel_Concrete.png" );
316             break;
317         case Background_Color::DIRT:
318             _background_texture->loadFromFile( "assets/images/PNG/GroundGravel_Dirt.png" );
319             break;
320         case Background_Color::GRASS:
321             _background_texture->loadFromFile( "assets/images/PNG/GroundGravel_Grass.png" );
322             break;
323         case Background_Color::SAND:
324             _background_texture->loadFromFile( "assets/images/PNG/GroundGravel_Sand.png" );
325             break;
```

```
326     }
327
328     min_within_enum = static_cast< int >( entity::Entity_Box::Color::BEIGE_LIGHT );
329     max_within_enum = static_cast< int >( entity::Entity_Box::Color::YELLOW_LIGHT );
330     std::uniform_int_distribution< int > box_distribution( min_within_enum, max_within_enum );
331     auto random_box_color = static_cast< entity::Entity_Box::Color >( box_distribution( mt ) );
332
333     min_within_enum = static_cast< int >( entity::Entity_Platform::Color::BEIGE );
334     max_within_enum = static_cast< int >( entity::Entity_Platform::Color::YELLOW );
335     std::uniform_int_distribution< int > platform_distribution( min_within_enum, max_within_enum );
336     auto random_platform_color = static_cast< entity::Entity_Platform::Color >( platform_distribution( mt )
        );
337
338     min_within_enum = static_cast< int >( entity::Entity_Wall::Color::BEIGE );
339     max_within_enum = static_cast< int >( entity::Entity_Wall::Color::BROWN );
340     std::uniform_int_distribution< int > wall_distribution( min_within_enum, max_within_enum );
341     auto random_wall_color = static_cast< entity::Entity_Wall::Color >( wall_distribution( mt ) );
342
343     int layers = 0;
344     sf::IntRect textureRect( _world_bounds );
345     _background_texture->setRepeated( true );
346
347     auto *backgroundSprite = new Sprite_Node( *_background_texture, textureRect );
348     backgroundSprite->setPosition( _world_bounds.left, _world_bounds.top );
349     _scene_layers.push_back( backgroundSprite );
350
351     layers++;
352
353     for ( model::Actor *actor: _board.get_world() )
354     {
355         float asset_coord_x;
356         float asset_coord_y;
357         float asset_coord_width;
358         float asset_coord_height;
359
360         sf::IntRect asset_rect;
361
362         Sprite_Node *actor_sprite = nullptr;
363
364         entity::Entity *entity_actor = nullptr;
365
366         if ( actor->get_type() == actor->PLAYER )
367         {
368             _board_player = dynamic_cast< model::Player * >( actor );
369             entity_actor = new entity::Entity_Player( actor->get_x(), actor->get_y() );
370             _player_entity = dynamic_cast< entity::Entity_Player * >( entity_actor );
371             auto player_asset_rect = _player_entity->get_player_face_map().find( entity::Entity_Player::Face
    ::SOUTH )->second;
372             asset_coord_x = player_asset_rect.at( 0 );
373             asset_coord_y = player_asset_rect.at( 1 );
374             asset_coord_width = player_asset_rect.at( 2 );
375             asset_coord_height = player_asset_rect.at( 3 );
376             asset_rect = sf::IntRect( asset_coord_x, asset_coord_y, asset_coord_width, asset_coord_height );
377             actor_sprite = new Sprite_Node( *_player_texture_sheet, asset_rect );
378             _player_sprite = actor_sprite;
379         }
380         if ( actor->get_type() == actor->PLATFORM )
381         {
382             entity_actor = new entity::Entity_Platform( actor->get_x(), actor->get_y() );
383             actor = dynamic_cast< entity::Entity_Platform * >( entity_actor );
384             auto *platform_actor = dynamic_cast< entity::Entity_Platform * >( actor );
385             auto platform_asset_rect = platform_actor->get_platform_color_map().find( random_platform_color
    )->second;
386             asset_coord_x = platform_asset_rect.at( 0 );
387             asset_coord_y = platform_asset_rect.at( 1 );
388             asset_coord_width = platform_asset_rect.at( 2 );
389             asset_coord_height = platform_asset_rect.at( 3 );
390             asset_rect = sf::IntRect( asset_coord_x, asset_coord_y, asset_coord_width, asset_coord_height );
391             actor_sprite = new Sprite_Node( *_platform_texture_sheet, asset_rect );
392             /* If sprite size = 32x32 */
393             actor_sprite->setOrigin(
394                     -( asset_coord_width / 2.f ),
395                     -( asset_coord_height / 2.f )
396             );
397         }
398         if ( actor->get_type() == actor->BOX )
399         {
400             _box_actors.push_back( dynamic_cast< model::Box * >( actor ) );
401             entity_actor = new entity::Entity_Box( actor->get_x(), actor->get_y() );
402             actor = dynamic_cast< entity::Entity_Box * >( entity_actor );
403             auto *box_entity = dynamic_cast< entity::Entity_Box * >( actor );
404             auto box_asset_rect = box_entity->get_box_color_map().find( random_box_color )->second;
405             asset_coord_x = box_asset_rect.at( 0 );
406             asset_coord_y = box_asset_rect.at( 1 );
```

```
407             asset_coord_width = box_asset_rect.at( 2 );
408             asset_coord_height = box_asset_rect.at( 3 );
409             sf::IntRect asset_rect( asset_coord_x, asset_coord_y, asset_coord_width, asset_coord_height );
410             actor_sprite = new Sprite_Node( *_box_texture_sheet, asset_rect );
411             _box_sprites.push_back( actor_sprite );
412             _box_entities.push_back( box_entity );
413         }
414         if ( actor->get_type() == actor->WALL )
415         {
416             entity_actor = new entity::Entity_Wall( actor->get_x(), actor->get_y() );
417             actor = dynamic_cast< entity::Entity_Wall * >( entity_actor );
418             auto *wall_actor = dynamic_cast< entity::Entity_Wall * >( actor );
419             auto wall_asset_rect = wall_actor->get_wall_color_map().find( random_wall_color )->second;
420             asset_coord_x = wall_asset_rect.at( 0 );
421             asset_coord_y = wall_asset_rect.at( 1 );
422             asset_coord_width = wall_asset_rect.at( 2 );
423             asset_coord_height = wall_asset_rect.at( 3 );
424             asset_rect = sf::IntRect( asset_coord_x, asset_coord_y, asset_coord_width, asset_coord_height );
425             actor_sprite = new Sprite_Node( *_wall_texture_sheet, asset_rect );
426         }
427         actor_sprite->setPosition( actor->get_x(), actor->get_y() );
428         _scene_layers.push_back( actor_sprite );
429         _entities.push_back( entity_actor );
430         layers++;
431     }
432 
433     Logger::log( LoggerLevel::INFO, "Number of layers loaded: " + std::to_string( layers ) );
434 }
435 
436 /**
437  * Check whether the board is completed
438  * @return whether the board is completed
439  */
440 bool World::is_board_completed() const
441 {
442     return _board.is_completed();
443 }
444 
445 /**
446  * Move the player up
447  * @param pressed Whether the up key is pressed
448  */
449 void World::move_up( bool pressed )
450 {
451     if( pressed )
452     {
453         steps_counter++;
454     }
455     _player_is_moving_up = pressed;
456 }
457 
458 /**
459  * Move the player down
460  * @param pressed Whether the down key is pressed
461  */
462 void World::move_down( bool pressed )
463 {
464     if( pressed )
465     {
466         steps_counter++;
467     }
468     _player_is_moving_down = pressed;
469 }
470 
471 /**
472  * Move the player left
473  * @param pressed Whether the left key is pressed
474  */
475 void World::move_left( bool pressed )
476 {
477     if( pressed )
478     {
479         steps_counter++;
480     }
481     _player_is_moving_left = pressed;
482 }
483 
484 /**
485  * Move the player right
486  * @param pressed Whether the right key is pressed
487  */
488 void World::move_right( bool pressed )
489 {
490     if( pressed )
```

```
491     {
492         steps_counter++;
493     }
494     _player_is_moving_right = pressed;
495 }
496
497 /**
498  * Setter for the number of resets done throughout the match
499  * @param reset_counter The current number of resets
500  */
501 void World::set_reset_counter( int reset_counter )
502 {
503     _reset_counter = reset_counter;
504 }
505
506 /**
507  * Getter for the number of resets called
508  * @return The number of resets
509  */
510 int World::get_reset_counter() const
511 {
512     return _reset_counter;
513 }
```

9.7 Components

9.7.1 Component.hpp

```
1  #ifndef SOKOBAN_COMPONENT_HPP
2  #define SOKOBAN_COMPONENT_HPP
3
4  #include <SFML/System/NonCopyable.hpp>
5  #include <SFML/Graphics/Drawable.hpp>
6  #include <SFML/Graphics/Transformable.hpp>
7
8  #include <memory>
9
10 namespace sf
11 {
12     class Event;
13 }
14
15 namespace sokoban
16 {
17     namespace ui
18     {
19         namespace gui
20         {
21             class Component
22             : public sf::Drawable
23             , public sf::Transformable
24             , private sf::NonCopyable
25                 {
26                 public:
27                     /** Generic declaration of a shared pointer meant to define a base nature to said pointer */
28                     typedef std::shared_ptr< Component > Ptr;
29                     Component();
30                     virtual ~Component();
31                     virtual bool is_selectable() const = 0;
32                     bool is_selected() const;
33                     virtual void select();
34                     virtual void deselect();
35                     virtual bool is_active() const;
36                     virtual void activate();
37                     virtual void deactivate();
38                     virtual void handle_event( const sf::Event &event ) = 0;
39                 private:
40                     bool _is_selected; /** Check whether the component is selected */
41                     bool _is_active; /** Check whether the component is active */
42                 };
43         }
44     }
45 }
46
47 #endif //SOKOBAN_COMPONENT_HPP
```

9.7.2 Component.cpp

```
1 #include "Component.hpp"
2
3 using namespace sokoban::ui::gui;
4
```

```
5  /**
6   * Component's initializer
7   * By default, it is neither selected nor active
8   */
9  Component::Component()
10 : _is_selected( false )
11 , _is_active( false )
12 {
13 }
14
15 /**
16  * Destructor with no specific action
17  */
18 Component::~Component()
19 = default;
20
21 /**
22  * Getter meant to retrieve whether a component is selected or not
23  */
24 bool Component::is_selected() const
25 {
26     return _is_selected;
27 }
28
29 /**
30  * Setter meant to define a component as selected
31  */
32 void Component::select()
33 {
34     _is_selected = true;
35 }
36
37 /**
38  * Setter meant to define a component as deselected
39  */
40 void Component::deselect()
41 {
42     _is_selected = false;
43 }
44
45 /**
46  * Getter meant to retrieve whether a component is active or not
47  */
48 bool Component::is_active() const
49 {
50     return _is_active;
51 }
52
53 /**
54  * Setter meant to define a component as active
55  */
56 void Component::activate()
57 {
58     _is_active = true;
59 }
60
61 /**
62  * Setter meant to define a component as non active
63  */
64 void Component::deactivate()
65 {
66     _is_active = false;
67 }
```

9.7.3 Container.hpp

```
1  #ifndef SOKOBAN_CONTAINER_HPP
2  #define SOKOBAN_CONTAINER_HPP
3
4  #include "Component.hpp"
5
6  #include <memory>
7  #include <vector>
8
9  namespace sokoban
10 {
11     namespace ui
12     {
13         namespace gui
14         {
15             class Container
16                     : public Component
17             {
```

```
18              public:
19                  /** Shared pointer meant to characterize the component as a Container */
20                  typedef std::shared_ptr< Container > Ptr;
21                  Container();
22                  void pack( Component::Ptr component );
23                  bool is_selectable() const override;
24                  void handle_event( const sf::Event &event ) override;
25              private:
26                  std::vector< Component::Ptr > _children; /** The various components it should hold */
27                  int _selected_child; /** The selected child */
28                  void draw( sf::RenderTarget &target, sf::RenderStates states ) const override;
29                  bool has_selection() const;
30                  void select( std::size_t index );
31                  void select_next();
32                  void select_previous();
33              };
34          }
35      }
36  }
37
38  #endif //SOKOBAN_CONTAINER_HPP
```

9.7.4 Container.cpp

```
1   #include "Container.hpp"
2
3   #include <SFML/Window/Event.hpp>
4   #include <SFML/Graphics/RenderStates.hpp>
5   #include <SFML/Graphics/RenderTarget.hpp>
6
7   using namespace sokoban::ui::gui;
8
9   /**
10   * Constructor for the container
11   * No children by default and no selected child by default
12   */
13  Container::Container()
14  : _children()
15  , _selected_child( -1 )
16  {
17  }
18
19  /**
20   * Function meant to add a component to the children.
21   * @param component The new child to add
22   */
23  void Container::pack( Component::Ptr component )
24  {
25      _children.push_back( component );
26      if( !has_selection() && component->is_selectable() )
27      {
28          select( _children.size() -1 );
29      }
30  }
31
32  /**
33   * By default a container is not selectable
34   */
35  bool Container::is_selectable() const
36  {
37      return false;
38  }
39
40  /**
41   * Function meant to interchange between the various children
42   */
43  void Container::handle_event( const sf::Event &event )
44  {
45      if( has_selection() && _children[ _selected_child ]->is_active() )
46      {
47          _children[ _selected_child ]->handle_event( event );
48      }
49      else if( event.type == sf::Event::KeyReleased )
50      {
51          if( event.key.code == sf::Keyboard:: Up )
52          {
53              select_previous();
54          }
55          else if( event.key.code == sf::Keyboard::Down )
56          {
57              select_next();
58          }
59          else if( event.key.code == sf::Keyboard::Return || event.key.code == sf::Keyboard::Space )
```

```
60          {
61              if( has_selection() )
62              {
63                  _children[ _selected_child ]->activate();
64              }
65          }
66      }
67  }
68
69  /**
70   * Visually display each child of the container
71   */
72  void Container::draw( sf::RenderTarget &target, sf::RenderStates states ) const
73  {
74      states.transform *= getTransform();
75      for( const Component::Ptr &child : _children )
76      {
77          target.draw( *child, states );
78      }
79  }
80
81  /**
82   * Function meant to retrieve whether a container has a selected child
83   */
84  bool Container::has_selection() const
85  {
86      return _selected_child >= 0;
87  }
88
89  /**
90   * Function meant to interchange between the different children
91   * @param index The child that has been selected
92   */
93  void Container::select( std::size_t index )
94  {
95      if( _children[ index ]->is_selectable() )
96      {
97          if( has_selection() )
98          {
99              _children[ _selected_child ]->deselect();
100         }
101         _children[ index ]->select();
102         _selected_child = index;
103     }
104 }
105
106 /**
107  * Function meant to select the following child within the list of children
108  */
109 void Container::select_next()
110 {
111     if( !has_selection() )
112     {
113         return;
114     }
115     int next = _selected_child;
116     do
117     {
118         next = ( next + 1 ) % _children.size();
119     }
120     while ( !_children[ next ]->is_selectable() );
121     select( next );
122 }
123
124 /**
125  * Function meant to select the previous child within the list of children
126  */
127 void Container::select_previous()
128 {
129     if( !has_selection() )
130     {
131         return;
132     }
133     int prev = _selected_child;
134     do
135     {
136         prev = ( prev + _children.size() - 1 ) % _children.size();
137     }
138     while ( !_children[ prev ]->is_selectable() );
139     select( prev );
140 }
```

9.7.5 Button.hpp

```
#ifndef SOKOBAN_BUTTON_HPP
#define SOKOBAN_BUTTON_HPP

#include "Component.hpp"
#include "../Resource_Identifiers.hpp"
#include "../states/State.hpp"

#include <SFML/Graphics/Sprite.hpp>
#include <SFML/Graphics/Text.hpp>

#include <memory>
#include <string>
#include <vector>
#include <functional>

namespace sokoban
{
    namespace ui
    {
        namespace gui
        {
            class Sound_Player;
            class Button
                    : public Component
            {
            public:
                typedef std::shared_ptr< Button > Ptr; /** The shared pointer that will define the Component
    's nature */
                typedef std::function< void() > Callback; /** The action the component will execute */
                enum Type
                {
                    Normal, /** Non-selected state of a button */
                    Selected, /** Selected state of a button */
                    Pressed, /** Pressed state of a button */
                    Button_Count /** Counter for various button states */
                };
                explicit Button( State::Context context );
                void set_callback( Callback callback );
                void set_text( const std::string &text );
                void set_toggle( bool flag );
                bool is_selectable() const override;
                void select() override;
                void deselect() override;
                void activate() override;
                void deactivate() override;
                void handle_event( const sf::Event &event ) override;
            private:
                Callback _callback; /** The type of execution the button will cause */
                sf::Sprite _sprite; /** The asset meant to represent the sprite */
                sf::Text _text; /** The text written on the button */
                bool _is_toggled; /** Whether the button is toggled or not */
                Sound_Player &_sounds; /** The sound it will make when selected or executed */
                void draw( sf::RenderTarget &target, sf::RenderStates states ) const override;
                void change_texture( Type button_type );
            };
        }
    }
}

#endif //SOKOBAN_BUTTON_HPP
```

9.7.6 Button.cpp

```
#include "Button.hpp"

#include "../Utility.hpp"
#include "../Sound_Player.hpp"
#include "../../Resource_Holder.hpp"

#include <SFML/Window/Event.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>

using namespace sokoban::ui::gui;

/**
 * Initializer for the button.
 * Set the default states that characterize the button
 */
Button::Button( State::Context context )
: _callback()
```

```
19 , _sprite( context._textures->get( Textures::Button ) )
20 , _text( "", context._fonts->get( Fonts::Rampart_One ) )
21 , _is_toggled( false )
22 , _sounds( *context._sounds )
23 {
24     change_texture( Normal );
25 
26     sf::FloatRect bounds = _sprite.getLocalBounds();
27     _text.setPosition( bounds.width / 2.f, bounds.height / 2.f );
28 }
29 
30 /**
31  * Setter for the type of action a button will execute
32  * @param callback The execution
33  */
34 void Button::set_callback( Button::Callback callback )
35 {
36     _callback = std::move( callback );
37 }
38 
39 /**
40  * Setter for the button's text
41  */
42 void Button::set_text( const std::string &text )
43 {
44     _text.setString( text );
45     Utility::center_origin( _text );
46 }
47 
48 /**
49  * Setter meant to make the button toggleable
50  * @param flag Whether it is toggleable or not
51  */
52 void Button::set_toggle( bool flag )
53 {
54     _is_toggled = flag;
55 }
56 
57 /**
58  * Button meant to determine whether a button is selectable or not.
59  */
60 bool Button::is_selectable() const
61 {
62     return true;
63 }
64 
65 /**
66  * Function meant to change the texture of a button upon selection
67  */
68 void Button::select()
69 {
70     Component::select();
71 
72     change_texture( Selected );
73     _sounds.play( Sound_Effect::Button_Click_02 );
74 }
75 
76 /**
77  * Function meant to deselect a component and change its texture
78  */
79 void Button::deselect()
80 {
81     Component::deselect();
82 
83     change_texture( Normal );
84 }
85 
86 /**
87  * Function meant to make a button active
88  */
89 void Button::activate()
90 {
91     Component::activate();
92     if( _is_toggled )
93     {
94         change_texture( Pressed );
95     }
96     if( _callback )
97     {
98         _callback();
99     }
100     if( !_is_toggled )
101     {
102         deactivate();
```

```
103     }
104     _sounds.play( Sound_Effect::Button_Click_01 );
105 }
106
107 /**
108  * Function meant to deactivate an already active button
109  */
110 void Button::deactivate()
111 {
112     Component::deactivate();
113
114     if( _is_toggled )
115     {
116         if( is_selected() )
117         {
118             change_texture( Selected );
119         }
120         else
121         {
122             change_texture( Normal );
123         }
124     }
125 }
126
127 /*
128  * Function meant to execute an event
129  */
130 void Button::handle_event( const sf::Event &event )
131 {
132 }
133
134 /**
135  * Function meant to draw the display the button graphically
136  * @param target The target meant to be displayed upon
137  * @param states The various preconfigured states that characterise the target
138  */
139 void Button::draw( sf::RenderTarget &target, sf::RenderStates states ) const
140 {
141     states.transform *= getTransform();
142     target.draw( _sprite, states );
143     target.draw( _text, states );
144 }
145
146 /**
147  * Function meant to change the texture of the button based on its current state
148  */
149 void Button::change_texture( Button::Type button_type )
150 {
151     sf::IntRect texture_rect( 0, 50 * button_type, 200, 50 );
152     _sprite.setTextureRect( texture_rect );
153 }
```

9.7.7 Label.hpp

```
1  #ifndef SOKOBAN_LABEL_HPP
2  #define SOKOBAN_LABEL_HPP
3
4  #include "Component.hpp"
5  #include "../Resource_Identifiers.hpp"
6  #include "../../Resource_Holder.hpp"
7
8  #include <SFML/Graphics/Text.hpp>
9
10 namespace sokoban
11 {
12     namespace ui
13     {
14         namespace gui
15         {
16             class Label
17                     : public Component
18             {
19             public:
20                 /** Shared pointer meant to define the component as a Label */
21                 typedef std::shared_ptr< Label > Ptr;
22                 Label( const std::string &text, const Font_Holder &fonts, float character_size );
23                 bool is_selectable() const override;
24                 void set_text( const std::string &text );
25                 void handle_event( const sf::Event &event ) override;
26             private:
27                 sf::Text _text; /** The text meant to be displayed */
28                 void draw( sf::RenderTarget &target, sf::RenderStates states ) const override;
29             };
```

```
30          }
31      }
32  }
33
34  #endif //SOKOBAN_LABEL_HPP
```

9.7.8 Label.cpp

```
1   #include "Label.hpp"
2
3   #include "../Utility.hpp"
4
5   #include <SFML/Graphics/RenderStates.hpp>
6   #include <SFML/Graphics/RenderTarget.hpp>
7
8   using namespace sokoban::ui::gui;
9
10  /**
11   * Constructor for the Label component
12   * @param text The text meant to be shown
13   * @param fonts The font it should be characterized in
14   * @param character_size The character size
15   */
16  Label::Label( const std::string &text, const sokoban::ui::Font_Holder &fonts, float character_size )
17  : _text( text, fonts.get( Fonts::Rampart_One ), character_size )
18  {
19  }
20
21  /**
22   * Make the label not selectable
23   * @return false since a label is just text
24   */
25  bool Label::is_selectable() const
26  {
27      return false;
28  }
29
30  /**
31   * Setter for the text in case of changes
32   * @param text The text meant to be shown
33   */
34  void Label::set_text( const std::string &text )
35  {
36      _text.setString( text );
37  }
38
39  /**
40   * Empty since a label does not execute any sort of action by default
41   * @param event None
42   */
43  void Label::handle_event( const sf::Event &event )
44  {
45  }
46
47  /**
48   * Function that will display the label visually
49   */
50  void Label::draw( sf::RenderTarget &target, sf::RenderStates states ) const
51  {
52      states.transform *= getTransform();
53      target.draw( _text, states );
54  }
```

9.8 Entities

9.8.1 Entity.hpp

```
1   #ifndef SOKOBAN_ENTITY_HPP
2   #define SOKOBAN_ENTITY_HPP
3
4   #include "../Scene_Node.hpp"
5
6   #include <map>
7   #include <array>
8   #include <string>
9   #include <ostream>
10
11  namespace sokoban
12  {
13      namespace ui
14      {
15          namespace gui
```

```
16          {
17              namespace entity
18              {
19                  class Entity
20                          : public Scene_Node
21                  {
22                  public:
23                      explicit Entity( std::array< float, 4 > asset_coords );
24                      Entity( const Entity &entity );
25                      Entity &operator=( const Entity &entity );
26                      ~Entity() override;
27                      std::array< float, 4 > get_asset_coords() const;
28                      void set_asset_coords( std::array< float, 4 > asset_coords );
29                      void update_current( sf::Time dt, Command_Queue &commands ) override;
30                  private:
31                      /** The asset's coordinates on the spritesheet */
32                      std::array< float, 4 > _assets_coords;
33                  };
34              }
35          }
36      }
37  }
38
39  #endif //SOKOBAN_ENTITY_HPP
```

9.8.2 Entity.cpp

```
1   #include "Entity.hpp"
2
3   #include <cassert>
4
5   using namespace sokoban::ui::gui::entity;
6
7   /**
8    * Entity constructor meant to initialize the asset's coordinates on the sprite sheet
9    */
10  Entity::Entity( std::array< float, 4 > asset_coords )
11  : _assets_coords( asset_coords )
12  {
13  }
14
15  /**
16   * Copy constructor for the entity
17   * @param entity The entity to copy the information from
18   */
19  Entity::Entity( const Entity &entity )
20  : Entity( entity._assets_coords )
21  {
22  }
23
24  /**
25   * Redefinition of the = operator
26   * @param entity The entity to copy the information from
27   * @return The new instance of the entity with the copied information
28   */
29  Entity &Entity::operator=( const Entity &entity )
30  {
31      if( &entity != this )
32      {
33          _assets_coords = entity._assets_coords;
34      }
35      return *this;
36  }
37
38  /**
39   * Default entity destructor
40   */
41  Entity::~Entity()
42  = default;
43
44  /**
45   * Getter for the assets coordinates
46   * @return Array of floats containing the asset's coordinates
47   */
48  std::array< float, 4 > Entity::get_asset_coords() const
49  {
50      return _assets_coords;
51  }
52
53  /**
54   * Setter for the asset's coordinates
55   * @param asset_coords The assets to retrieve and set as new
56   */
```

```
57 void Entity::set_asset_coords( std::array< float, 4 > asset_coords )
58 {
59     _assets_coords = asset_coords;
60 }
61
62 /**
63  * Function to update an Entity graphically based on its action
64  * @param dt The clock time
65  * @param commands The commands to execute
66  */
67 void Entity::update_current( sf::Time dt, Command_Queue &commands )
68 {
69 }
```

9.8.3 Entity_Box.hpp

```
1  #ifndef SOKOBAN_ENTITY_BOX_HPP
2  #define SOKOBAN_ENTITY_BOX_HPP
3
4  #include "../../../model/Box.hpp"
5  #include "Entity_Movable.hpp"
6
7  #include <map>
8  #include <array>
9  #include <string>
10 #include <ostream>
11
12 namespace sokoban
13 {
14     namespace ui
15     {
16         namespace gui
17         {
18             namespace entity
19             {
20
21                 class Entity_Box
22                           : public model::Box
23                             , public Entity_Movable
24                 {
25
26                 public:
27                     /** The various colors of a box */
28                     enum class Color
29                     {
30                         BEIGE_LIGHT,
31                         BLACK_LIGHT,
32                         BLUE_LIGHT,
33                         BROWN_LIGHT,
34                         BEIGE_DARK,
35                         BLACK_DARK,
36                         BLUE_DARK,
37                         BROWN_DARK,
38                         WHITE_DARK,
39                         PURPLE_DARK,
40                         RED_DARK,
41                         YELLOW_DARK,
42                         WHITE_LIGHT,
43                         PURPLE_LIGHT,
44                         RED_LIGHT,
45                         YELLOW_LIGHT
46                     };
47                     Entity_Box( float x, float y );
48                     Entity_Box( const Entity_Box &box );
49                     Entity_Box &operator=( const Entity_Box &box );
50                     ~Entity_Box() override;
51                     ID get_type() const override;
52                     std::string to_string() const override;
53                     friend std::ostream &operator<<( std::ostream &os, const Entity_Box &box );
54                     const std::map< Color, std::array< float, 4>> &get_box_color_map() const;
55                 private:
56                     /** Mapper containing the coordinates of each asset based on the color */
57                     std::map< Color, std::array< float, 4 > > _box_color_map;
58                 };
59             }
60         }
61     }
62 }
63
64 #endif //SOKOBAN_ENTITY_BOX_HPP
```

9.8.4 Entity_Box.cpp

```cpp
#include "Entity_Box.hpp"

using namespace sokoban::ui::gui::entity;
using namespace sokoban::model;

const float OFFSET = 64;

const std::array< float, 4 > box_beige_light_asset = {
        0
        , 0
        , OFFSET
        , OFFSET
};
const std::array< float, 4 > box_black_light_asset = {
        OFFSET * 1
        , 0
        , OFFSET
        , OFFSET
};
const std::array< float, 4 > box_blue_light_asset = {
        OFFSET * 2
        , 0
        , OFFSET
        , OFFSET
};
const std::array< float, 4 > box_brown_light_asset = {
        OFFSET * 3
        , 0
        , OFFSET
        , OFFSET
}; /** The default asset for the box */
const std::array< float, 4 > box_beige_dark_asset = {
        OFFSET * 4
        , 0
        , OFFSET
        , OFFSET
};
const std::array< float, 4 > box_black_dark_asset = {
        OFFSET * 5
        , 0
        , OFFSET
        , OFFSET
};
const std::array< float, 4 > box_blue_dark_asset = {
        OFFSET * 6
        , 0
        , OFFSET
        , OFFSET
};
const std::array< float, 4 > box_brown_dark_asset = {
        OFFSET * 7
        , 0
        , OFFSET
        , OFFSET
};
const std::array< float, 4 > box_white_dark_asset = {
        OFFSET * 8
        , 0
        , OFFSET
        , OFFSET
};
const std::array< float, 4 > box_purple_dark_asset = {
        OFFSET * 9
        , 0
        , OFFSET
        , OFFSET
};
const std::array< float, 4 > box_red_dark_asset = {
        OFFSET * 10
        , 0
        , OFFSET
        , OFFSET
};
const std::array< float, 4 > box_yellow_dark_asset = {
        OFFSET * 11
        , 0
        , OFFSET
        , OFFSET
};
const std::array< float, 4 > box_white_light_asset = {
        OFFSET * 12
        , 0
        , OFFSET
        , OFFSET
```

```
85 };
86 const std::array< float, 4 > box_purple_light_asset = {
87         OFFSET * 13
88         , 0
89         , OFFSET
90         , OFFSET
91 };
92 const std::array< float, 4 > box_red_light_asset = {
93         OFFSET * 14
94         , 0
95         , OFFSET
96         , OFFSET
97 };
98 const std::array< float, 4 > box_yellow_light_asset = {
99         OFFSET * 15
100        , 0
101        , OFFSET
102        , OFFSET
103 };
104
105 /**
106  * Constructor for the entity box
107  * @param x The position on the X axis
108  * @param y The position on the Y axis
109  */
110 Entity_Box::Entity_Box( float x, float y )
111         : model::Box( x, y )
112         , Entity_Movable( box_brown_light_asset )
113 {
114
115     _box_color_map = std::map< Color, std::array< float, 4 > >
116             {
117                     {   Color::BEIGE_LIGHT , box_beige_light_asset }
118                     , { Color::BLACK_LIGHT , box_black_light_asset }
119                     , { Color::BLUE_LIGHT  , box_blue_light_asset }
120                     , { Color::BROWN_LIGHT , box_brown_light_asset }
121                     , { Color::BEIGE_DARK  , box_beige_dark_asset }
122                     , { Color::BLACK_DARK  , box_black_dark_asset }
123                     , { Color::BLUE_DARK   , box_blue_dark_asset }
124                     , { Color::BROWN_DARK  , box_brown_dark_asset }
125                     , { Color::WHITE_DARK  , box_white_dark_asset }
126                     , { Color::PURPLE_DARK , box_purple_dark_asset }
127                     , { Color::RED_DARK    , box_red_dark_asset }
128                     , { Color::YELLOW_DARK , box_yellow_dark_asset }
129                     , { Color::WHITE_LIGHT , box_white_light_asset }
130                     , { Color::PURPLE_LIGHT, box_purple_light_asset }
131                     , { Color::RED_LIGHT   , box_red_light_asset }
132                     , { Color::YELLOW_LIGHT, box_yellow_light_asset }
133             };
134 }
135
136 /**
137  * Copy constructor of a box entity
138  */
139 Entity_Box::Entity_Box( const Entity_Box &box )
140         : Entity_Box( box.get_x(), box.get_y() )
141 {
142
143 }
144
145 /**
146  * Redefinition of the = operator
147  * @param box The box we wish to copy the information from
148  * @return The new box with the copied box's information
149  */
150 Entity_Box &Entity_Box::operator=( const Entity_Box &box )
151 {
152     if ( &box != this )
153     {
154         set_asset_coords( box.get_asset_coords() );
155     }
156     return *this;
157 }
158
159 /**
160  * Based destructor for the box entity
161  */
162 Entity_Box::~Entity_Box()
163 = default;
164
165 /**
166  * Getter meant to retrieve the actor's nature
167  */
168 Actor::ID Entity_Box::get_type() const
```

```
169 {
170     return Actor::BOX;
171 }
172 
173 /**
174  * Textual format of the box's information
175  * @return The Box Entity's information in textual format
176  */
177 std::string Entity_Box::to_string() const
178 {
179     return Box::to_string();
180 }
181 
182 /**
183  * Getter for the mapper containing the various coordinates of the box based on the color
184  * @return The color's mapper
185  */
186 const std::map< Entity_Box::Color, std::array< float, 4>> &Entity_Box::get_box_color_map() const
187 {
188     return _box_color_map;
189 }
190 
191 /**
192  * Redefinition of the << operator meant to output the box directly within an std::cout
193  * @param os The output stream to send the text to
194  * @param box The box we wish to output
195  * @return Textual output containing the Entity's information
196  */
197 std::ostream &sokoban::ui::gui::entity::operator<<( std::ostream &os, const Entity_Box &box )
198 {
199     os << box.to_string();
200     return os;
201 }
```

9.8.5 Entity_Movable.hpp

```
1  #ifndef SOKOBAN_ENTITY_MOVABLE_HPP
2  #define SOKOBAN_ENTITY_MOVABLE_HPP
3  
4  #include "Entity.hpp"
5  
6  #include <map>
7  #include <array>
8  #include <string>
9  #include <ostream>
10 
11 namespace sokoban
12 {
13     namespace ui
14     {
15         namespace gui
16         {
17             namespace entity
18             {
19                 class Entity_Movable
20                         : public Entity
21                 {
22                 public:
23                     explicit Entity_Movable( std::array< float, 4 > asset_coords );
24                     void set_direction( float dx, float dy );
25                     sf::Vector2f get_direction() const;
26                     void update_current( sf::Time dt, Command_Queue &commands ) override;
27                 private:
28                     /** The direction the Entity has to move to */
29                     sf::Vector2f _direction;
30                 };
31             }
32         }
33     }
34 }
35 
36 #endif //SOKOBAN_ENTITY_MOVABLE_HPP
```

9.8.6 Entity_Movable.cpp

```
1 #include "Entity_Movable.hpp"
2 
3 using namespace sokoban::ui::gui::entity;
4 
5 /**
6  * Constructor for a movable entity
7  * @param asset_coords The asset's coordinates
```

```
 */
Entity_Movable::Entity_Movable( std::array< float, 4 > asset_coords )
        : Entity( asset_coords )
        , _direction()
{
}

/**
 * The direction where the entity moves to
 * @param dx Movement distance of the X axis
 * @param dy Movement distance of the Y axis
 */
void Entity_Movable::set_direction( float dx, float dy )
{
    _direction.x = dx;
    _direction.y = dy;
}

/**
 * Getter for the direction where the entity moves to
 * @return The vector containing the X and Y axis of the direction
 */
sf::Vector2f Entity_Movable::get_direction() const
{
    return _direction;
}

/**
 * Visual real-time update of a movable entity along with its commands
 */
void Entity_Movable::update_current( sf::Time dt, Command_Queue &commands )
{
    Entity_Movable::move( _direction * dt.asSeconds() );
}
```

9.8.7 Entity_Platform.hpp

```
#ifndef SOKOBAN_ENTITY_PLATFORM_HPP
#define SOKOBAN_ENTITY_PLATFORM_HPP

#include "../../../model/Platform.hpp"
#include "Entity.hpp"

#include <map>
#include <array>
#include <string>
#include <ostream>

namespace sokoban
{
    namespace ui
    {
        namespace gui
        {
            namespace entity
            {
                class Entity_Platform
                        : public model::Platform
                        , public Entity
                {
                public:
                    enum class Color
                    {
                        BEIGE,
                        BLACK,
                        BLUE,
                        BROWN,
                        WHITE,
                        PURPLE,
                        RED,
                        YELLOW
                    };
                    Entity_Platform( float x, float y );
                    Entity_Platform( const Entity_Platform &entity );
                    Entity_Platform &operator=( const Entity_Platform &entity );
                    ~Entity_Platform() override;
                    std::string to_string() const;
                    const std::map< Color, std::array< float, 4>> &get_platform_color_map() const;
                    friend std::ostream &operator<<( std::ostream &os, const Entity_Platform &entity );
                private:
                    /** Mapper containing the coordinates of each asset based on the color */
                    std::map< Color, std::array< float, 4 > > _platform_color_map;
                };
```

```
47              }
48          }
49      }
50  }
51
52  #endif //SOKOBAN_ENTITY_PLATFORM_HPP
```

9.8.8 Entity_Platform.cpp

```
1   #include "Entity_Platform.hpp"
2
3   using namespace sokoban::ui::gui::entity;
4
5   const float OFFSET = 32;
6
7   const std::array< float, 4 > platform_beige_asset = {
8           0
9           , 0
10          , OFFSET
11          , OFFSET
12  };
13
14  const std::array< float, 4 > platform_black_asset = {
15          OFFSET * 1
16          , 0
17          , OFFSET
18          , OFFSET
19  };
20
21  const std::array< float, 4 > platform_blue_asset = {
22          OFFSET * 2
23          , 0
24          , OFFSET
25          , OFFSET
26  };
27
28  const std::array< float, 4 > platform_brown_asset = {
29          OFFSET * 3
30          , 0
31          , OFFSET
32          , OFFSET
33  };
34
35  const std::array< float, 4 > platform_white_asset = {
36          OFFSET * 4
37          , 0
38          , OFFSET
39          , OFFSET
40  };
41
42  const std::array< float, 4 > platform_purple_asset = {
43          OFFSET * 5
44          , 0
45          , OFFSET
46          , OFFSET
47  };
48
49  const std::array< float, 4 > platform_red_asset = {
50          OFFSET * 6
51          , 0
52          , OFFSET
53          , OFFSET
54  };
55
56  const std::array< float, 4 > platform_yellow_asset = {
57          OFFSET * 7
58          , 0
59          , OFFSET
60          , OFFSET
61  };
62
63  /**
64   * Default Entity Platform constructor
65   * @param x The coordinates on the X axis
66   * @param y The coordinates on the Y axis
67   */
68  Entity_Platform::Entity_Platform( float x, float y )
69  : Platform( x, y )
70  , Entity( platform_blue_asset )
71  {
72      _platform_color_map = std::map< Color, std::array< float, 4 > >
73              {
74                      {   Color::BEIGE , platform_beige_asset }
```

```
75                      , { Color::BLACK , platform_black_asset }
76                      , { Color::BLUE  , platform_blue_asset }
77                      , { Color::BROWN , platform_brown_asset }
78                      , { Color::WHITE , platform_white_asset }
79                      , { Color::PURPLE, platform_purple_asset }
80                      , { Color::RED   , platform_red_asset }
81                      , { Color::YELLOW, platform_yellow_asset }
82              };
83
84  }
85
86  /**
87   * The copy constructor for the Entity Platform
88   * @param entity The Platform we wish to copy the information from
89   */
90  Entity_Platform::Entity_Platform( const Entity_Platform &entity )
91  : Entity_Platform( entity.get_x(), entity.get_y() )
92  {
93  }
94
95  /**
96   * Redefinition of the = operator
97   * @param entity The entity we wish to copy the information from
98   * @return The entity with the copied platform's information
99   */
100 Entity_Platform &Entity_Platform::operator=( const Entity_Platform &entity )
101 {
102     if( &entity != this )
103     {
104         model::Platform::operator=( entity );
105         Entity::operator=( entity );
106     }
107     return *this;
108 }
109
110 /**
111  * Default constructor for the Platform entity
112  */
113 Entity_Platform::~Entity_Platform()
114 = default;
115
116 /**
117  * Textual output containing the platform's information
118  * @return The textual output with the platform's information
119  */
120 std::string Entity_Platform::to_string() const
121 {
122     return model::Platform::to_string();
123 }
124
125 /**
126  * Getter for the mapper containing the asset's coordinates based on the color
127  * @return The mapper containing the coordinates based on the color
128  */
129 const std::map< Entity_Platform::Color, std::array< float, 4>> &Entity_Platform::get_platform_color_map()
        const
130 {
131     return _platform_color_map;
132 }
133
134 /**
135  * Redefinition of the << operator
136  * @param os The output stream to display the text to
137  * @param entity The entity we wish to output
138  * @return The textual output containing the entity's information
139  */
140 std::ostream &sokoban::ui::gui::entity::operator<<( std::ostream &os, const Entity_Platform &entity )
141 {
142     os << entity.to_string();
143     return os;
144 }
```

9.8.9 Entity_Player.hpp

```
1  #ifndef SOKOBAN_ENTITY_PLAYER_HPP
2  #define SOKOBAN_ENTITY_PLAYER_HPP
3
4  #include "../../../model/Player.hpp"
5  #include "Entity_Movable.hpp"
6
7  #include <SFML/Window/Event.hpp>
8
9  #include <map>
```

```
10 #include <array>
11 #include <string>
12 #include <ostream>
13
14 namespace sokoban
15 {
16     namespace ui
17     {
18         namespace gui
19         {
20             namespace entity
21             {
22                 class Entity_Player
23                         : public model::Player
24                           , public Entity_Movable
25                 {
26                 public:
27                     enum class Face
28                     {
29                         NORTH,
30                         SOUTH,
31                         EAST,
32                         WEST
33                     };
34                     Entity_Player( float x, float y );
35                     Entity_Player( const Entity_Player &entity );
36                     Entity_Player &operator=( const Entity_Player &entity );
37                     ~Entity_Player() override;
38                     std::string to_string() const override;
39                     const std::map< Face, std::array< float, 4>> &get_player_face_map() const;
40                     friend std::ostream &operator<<( std::ostream &os, const Entity_Player &entity );
41                 private:
42                     /** Mapper containing the coordinates of each asset based on the direction the player is
       looking */
43                     std::map< Face, std::array< float, 4 > > _player_face_map;
44                 };
45             }
46         }
47     }
48 }
49
50 #endif //SOKOBAN_ENTITY_PLAYER_HPP
```

9.8.10 Entity_Player.cpp

```
1  #include "Entity_Player.hpp"
2
3  using namespace sokoban::ui::gui::entity;
4
5  const float OFFSET = 64;
6
7  std::array< float, 4 > character_facing_west = {
8          0
9          , 0
10         , OFFSET
11         , OFFSET
12 };
13 std::array< float, 4 > character_walking_west = {
14         OFFSET * 1
15         , 0
16         , OFFSET
17         , OFFSET
18 };
19
20 std::array< float, 4 > character_facing_east = {
21         OFFSET * 2
22         , 0
23         , OFFSET
24         , OFFSET
25 };
26 std::array< float, 4 > character_walking_east = {
27         OFFSET * 3
28         , 0
29         , OFFSET
30         , OFFSET
31 };
32
33 std::array< float, 4 > character_facing_south = {
34         OFFSET * 4
35         , 0
36         , OFFSET
37         , OFFSET
38 };
```

```
39
40 std::array< float, 4 > character_walking_south_right_step = {
41         OFFSET * 5
42         , 0
43         , OFFSET
44         , OFFSET
45 };
46 std::array< float, 4 > character_walking_south_left_step = {
47         OFFSET * 6
48         , 0
49         , OFFSET
50         , OFFSET
51 };
52
53 std::array< float, 4 > character_facing_north = {
54         OFFSET * 7
55         , 0
56         , OFFSET
57         , OFFSET
58 };
59 std::array< float, 4 > character_walking_north_right_step = {
60         OFFSET * 8
61         , 0
62         , OFFSET
63         , OFFSET
64 };
65 std::array< float, 4 > character_walking_north_left_step = {
66         OFFSET * 9
67         , 0
68         , OFFSET
69         , OFFSET
70 };
71
72 const std::array< float, 4 > player_asset_north = character_facing_north;
73 const std::array< float, 4 > player_asset_south = character_facing_south;
74 const std::array< float, 4 > player_asset_east = character_facing_east;
75 const std::array< float, 4 > player_asset_west = character_facing_west;
76
77 /**
78  * Default constructor for the Player entity
79  * @param x The coordinates on the X axis
80  * @param y The coordinates on the Y axis
81  */
82 Entity_Player::Entity_Player( float x, float y )
83         : model::Player( x, y )
84         , Entity_Movable( player_asset_south )
85 {
86     _player_face_map = std::map< Face, std::array< float, 4 > >
87             {
88                     {   Face::NORTH, player_asset_north }
89                     , { Face::SOUTH, player_asset_south }
90                     , { Face::EAST , player_asset_east }
91                     , { Face::WEST , player_asset_west }
92             };
93 }
94
95 /**
96  * Copy constructor for the player entity
97  * @param entity The entity we wish to get the information from
98  */
99 Entity_Player::Entity_Player( const Entity_Player &entity )
100 : Entity_Player( entity.get_x(), entity.get_y() )
101 {
102     set_asset_coords( entity.get_asset_coords() );
103     _player_face_map = entity._player_face_map;
104 }
105
106 /**
107  * Redefinition of the = operator
108  * @param entity The entity we wish to get the information from
109  * @return The new instance of a place with the copied information
110  */
111 Entity_Player &Entity_Player::operator=( const Entity_Player &entity )
112 {
113     if( &entity != this )
114     {
115         model::Player::operator=( entity );
116         Entity::operator=( entity );
117         set_asset_coords( entity.get_asset_coords() );
118         _player_face_map = entity._player_face_map;
119     }
120     return *this;
121 }
122
```

```
123 /**
124  * Default destructor for the entity player
125  */
126 Entity_Player::~Entity_Player()
127 = default;
128
129 /**
130  * Textual output containing the Player's information
131  * @return Text containing the player's information
132  */
133 std::string Entity_Player::to_string() const
134 {
135     return Player::to_string();
136 }
137
138 /**
139  * Getter for the mapper containing the coordinates of each asset based on where the player is looking
140  * @return The mapper containing the assets
141  */
142 const std::map< Entity_Player::Face, std::array< float, 4>> &Entity_Player::get_player_face_map() const
143 {
144     return _player_face_map;
145 }
146
147 /**
148  * Redefinition of the << operator
149  * @param os The output we wish to inject the text into
150  * @param entity The entity we wish to output
151  * @return Textual output displaying the player's information
152  */
153 std::ostream &sokoban::ui::gui::entity::operator<<( std::ostream &os, const Entity_Player &entity )
154 {
155     os << entity.to_string();
156     return os;
157 }
```

9.8.11 Entity_Wall.hpp

```
1  #ifndef SOKOBAN_ENTITY_WALL_HPP
2  #define SOKOBAN_ENTITY_WALL_HPP
3
4  #include "../../../model/Wall.hpp"
5  #include "Entity.hpp"
6
7  #include <map>
8  #include <array>
9  #include <string>
10 #include <ostream>
11
12 namespace sokoban
13 {
14     namespace ui
15     {
16         namespace gui
17         {
18             namespace entity
19             {
20                 class Entity_Wall
21                         : public model::Wall
22                           , public Entity
23                 {
24                 public:
25                     enum class Color
26                     {
27                         BEIGE,
28                         BLACK,
29                         BROWN,
30                         WHITE
31                     };
32                     Entity_Wall( float x, float y );
33                     Entity_Wall( const Entity_Wall &entity );
34                     Entity_Wall &operator=( const Entity_Wall &entity );
35                     ~Entity_Wall() override;
36                     std::string to_string() const;
37                     const std::map< Color, std::array< float, 4>> &get_wall_color_map() const;
38                     friend std::ostream &operator<<( std::ostream &os, const Entity_Wall &entity );
39                 private:
40                     /** Mapper containing the coordinates of each asset based on the color */
41                     std::map< Color, std::array< float, 4 > > _wall_color_map;
42                 };
43             }
44         }
45     }
```

```
}

#endif //SOKOBAN_ENTITY_WALL_HPP
```

9.8.12 Entity_Wall.cpp

```
#include "Entity_Wall.hpp"

using namespace sokoban::ui::gui::entity;

const float OFFSET = 64;

const std::array< float, 4 > wall_round_beige_asset = {
        0
        , 0
        , OFFSET
        , OFFSET
};

const std::array< float, 4 > wall_round_black_asset = {
        OFFSET * 1
        , 0
        , OFFSET
        , OFFSET
};

const std::array< float, 4 > wall_round_brown_asset = {
        OFFSET * 2
        , 0
        , OFFSET
        , OFFSET
};

const std::array< float, 4 > wall_round_white_asset = {
        OFFSET * 3
        , 0
        , OFFSET
        , OFFSET
};

/**
 * Default constructor for the Wall Entity
 * @param x The coordinates on the X axis
 * @param y The coordinates on the Y axis
 */
Entity_Wall::Entity_Wall( float x, float y )
        : model::Wall( x, y )
        , Entity( wall_round_white_asset )
{
    _wall_color_map = std::map< Color, std::array< float, 4 > >
            {
                    {   Color::BEIGE, wall_round_beige_asset }
                    , { Color::BLACK, wall_round_black_asset }
                    , { Color::BROWN, wall_round_brown_asset }
                    , { Color::WHITE, wall_round_white_asset }
            };
}

/**
 * Copy constructor for the Wall entity
 * @param entity The entity we wish to copy the information from
 */
Entity_Wall::Entity_Wall( const Entity_Wall &entity )
: Entity_Wall( entity.get_x(), entity.get_y() )
{
}

/**
 * Redefinition of the = operator
 * @param entity The entity we wish to copy the information from
 * @return The new entity with the copied information
 */
Entity_Wall &Entity_Wall::operator=( const Entity_Wall &entity )
{
    if( &entity != this )
    {
        model::Wall::operator=( entity );
        Entity::operator=( entity );
    }
    return *this;
}

/**
```

```
78  * Default destructor for the Wall entity
79  */
80 Entity_Wall::~Entity_Wall()
81 = default;
82
83 /**
84  * Textual output of the Wall entity
85  * @return Textual output containing the Wall's information
86  */
87 std::string Entity_Wall::to_string() const
88 {
89     return Wall::to_string();
90 }
91
92 /**
93  * Getter for the mapper containing the coordinates of the assets based on each color
94  * @return The mapper containing the coordinates of the assets based on each color
95  */
96 const std::map< Entity_Wall::Color, std::array< float, 4>> &Entity_Wall::get_wall_color_map() const
97 {
98     return _wall_color_map;
99 }
100
101 /**
102  * Redefinition of the << operator
103  * @param os The output we wish to inject our information to
104  * @param entity The entity we wish to get the information from
105  * @return The textual output containing the entity's information
106  */
107 std::ostream &sokoban::ui::gui::entity::operator<<( std::ostream &os, const Entity_Wall &entity )
108 {
109     os << entity.to_string();
110     return os;
111 }
```

9.9 States

9.9.1 State_Identifiers.hpp

```
1  #ifndef SOKOBAN_STATE_IDENTIFIERS_HPP
2  #define SOKOBAN_STATE_IDENTIFIERS_HPP
3
4  namespace sokoban
5  {
6      namespace ui
7      {
8          namespace gui
9          {
10             namespace States
11             {
12                 /**
13                  * ID of each state that characterizes it
14                  */
15                 enum ID
16                 {
17                     None, /** No nature whatsoever */
18                     Title, /** The title screen */
19                     Menu, /** The menu screen */
20                     Game, /** The playable area */
21                     Pause, /** The pause screen */
22                     Settings, /** The settings screen */
23                 };
24             }
25         }
26     }
27 }
28
29 #endif //SOKOBAN_STATE_IDENTIFIERS_HPP
```

9.9.2 State.hpp

```
1  #pragma once
2  #ifndef SOKOBAN_STATE_HPP
3  #define SOKOBAN_STATE_HPP
4
5  #include "State_Identifiers.hpp"
6  #include "../Resource_Identifiers.hpp"
7
8  #include <SFML/System/Time.hpp>
9  #include <SFML/Window/Event.hpp>
10
11 #include <memory>
```

```
12
13 /**
14  * Pre declaration of the SFML RenderWindow class
15  */
16 namespace sf
17 {
18     class RenderWindow;
19 }
20
21 namespace sokoban
22 {
23     namespace ui
24     {
25         namespace gui
26         {
27             class State_Stack; /** The stack containing the various states */
28             class Music_Player; /** The global music player */
29             class Sound_Player; /** The global sound effects player */
30             class State
31             {
32             public:
33                 /** Unique pointer that characterizes the base nature of a state */
34                 typedef std::unique_ptr< State > Ptr;
35
36                 /**
37                  * Public struct containing globally accessible variables
38                  */
39                 struct Context
40                 {
41                     Context( sf::RenderWindow &window, Texture_Holder &textures, Font_Holder &fonts,
     Music_Player &music, Sound_Player &sounds );
42                     sf::RenderWindow *_window;
43                     Texture_Holder *_textures;
44                     Font_Holder *_fonts;
45                     Music_Player *_music;
46                     Sound_Player *_sounds;
47                 };
48
49                 State( State_Stack &stack, Context context );
50                 virtual ~State();
51                 virtual void draw() = 0;
52                 virtual bool update( sf::Time dt ) = 0;
53                 virtual bool handle_event( const sf::Event &event ) = 0;
54             protected:
55                 void request_stack_push( States::ID stateID );
56                 void request_stack_pop();
57                 void request_state_clear();
58                 Context get_context() const;
59             private:
60                 State_Stack *_stack; /** The stack containing the various states */
61                 Context _context; /** The globally accessible context */
62             };
63         }
64     }
65 }
66
67 #endif //SOKOBAN_STATE_HPP
```

9.9.3 State.cpp

```
1 #include "State.hpp"
2
3 #include "State_Stack.hpp"
4
5 using namespace sokoban::ui::gui;
6
7 /**
8  * Default constructor for the context
9  * @param window The window where the entities and components have to be displayed on
10  * @param textures The various textures that characterize each entity and component
11  * @param fonts The various fonts used throughout the usage of the software
12  * @param music The various songs that can be heard
13  * @param sounds The various sound effects played
14  */
15 State::Context::Context( sf::RenderWindow &window, Texture_Holder &textures, Font_Holder &fonts,
     Music_Player &music, Sound_Player &sounds )
16         : _window( &window )
17           , _textures( &textures )
18           , _fonts( &fonts )
19           , _music( &music )
20           , _sounds( &sounds )
21 {
22 }
```

```
23
24 /**
25  * Default constructor for a state
26  * @param stack The stack containing the state
27  * @param context The context that can be accessed throughout each state
28  */
29 State::State( State_Stack &stack, State::Context context )
30         : _stack( &stack )
31           , _context( context )
32 {
33 }
34
35 /**
36  * Default state destructor
37  */
38 State::~State()
39 = default;
40
41 /**
42  * Function meant to insert a state inside the stack based on its ID
43  * @param stateID The unique ID of the state
44  */
45 void State::request_stack_push( States::ID stateID )
46 {
47     _stack->push_state( stateID );
48 }
49
50 /**
51  * Function meant to remove a state from the stack
52  */
53 void State::request_stack_pop()
54 {
55     _stack->pop_state();
56 }
57
58 /**
59  * Stack meant to clear out completely the stack
60  */
61 void State::request_state_clear()
62 {
63     _stack->clear_states();
64 }
65
66 /**
67  * Getter for the context accessed all around the states
68  * @return The context to which we can access the various resources
69  */
70 State::Context State::get_context() const
71 {
72     return _context;
73 }
```

9.9.4 State_Game.hpp

```
1  #ifndef SOKOBAN_STATE_GAME_HPP
2  #define SOKOBAN_STATE_GAME_HPP
3
4  #include "State.hpp"
5  #include "../World.hpp"
6
7  #include <SFML/Graphics/Text.hpp>
8  #include <SFML/Graphics/Sprite.hpp>
9
10 namespace sokoban
11 {
12     namespace ui
13     {
14         namespace gui
15         {
16             class State_Game
17                     : public State
18             {
19             public:
20                 State_Game( State_Stack &stack, Context context );
21                 ~State_Game() override;
22                 bool update( sf::Time dt ) override;
23                 void draw() override;
24                 bool handle_event( const sf::Event &event ) override;
25             private:
26                 World *_world; /** The map that has to be displayed */
27                 sf::RenderWindow &_window; /** The window where everything has to be drawn into */
28                 std::vector< std::string > _levels; /** The various levels throughout the game */
29                 std::string _level; /** The level that is currently being played */
```

```
30                 sf::Text _text; /** The textual information present on the screen */
31                 void reset_board();
32                 void next_level();
33                 void prev_level();
34             };
35         }
36     }
37 }
38
39 #endif //SOKOBAN_STATE_GAME_HPP
```

9.9.5 State_Game.cpp

```
1  #include "State_Game.hpp"
2
3  #include "../../../util/Logger.hpp"
4
5  #include "../Music_Player.hpp"
6  #include "../Utility.hpp"
7
8  #include <SFML/Graphics/RenderWindow.hpp>
9
10 #include <boost/filesystem.hpp>
11
12 using namespace sokoban::ui::gui;
13 using namespace sokoban::util;
14
15
16 namespace
17 {
18     int current_level;
19     int reset_counter = 0;
20 }
21
22 /**
23  * Alphabetical sorting algorithm for the various paths found
24  * @param a A first path to sort
25  * @param b A second path to sort with
26  * @return The paths in the right order
27  */
28 bool sort_alphabetically( const boost::filesystem::path &a, const boost::filesystem::path &b )
29 {
30     return a.string() < b.string();
31 }
32
33 /**
34  * Getter for the various levels that have been detected inside the folder
35  */
36 std::vector< boost::filesystem::path > get_all_levels()
37 {
38     const boost::filesystem::path root = "assets/levels";
39     const std::string extension = ".lvl";
40     std::vector< boost::filesystem::path > paths;
41     if ( boost::filesystem::exists( root ) && boost::filesystem::is_directory( root ) )
42     {
43         for ( auto const &entry: boost::filesystem::recursive_directory_iterator( root ) )
44         {
45             if ( boost::filesystem::is_regular_file( entry ) )
46             {
47                 paths.emplace_back( entry );
48             }
49         }
50     }
51
52     std::sort( paths.begin(), paths.end(), sort_alphabetically );
53
54     return paths;
55 }
56
57 /**
58  * Default constructor for the Game State
59  * @param stack The stack containing the various states
60  * @param context The context containing the various resources
61  */
62 State_Game::State_Game( State_Stack &stack, Context context )
63         : State( stack, context )
64         , _window( *context._window )
65           , _level()
66           , _levels()
67           , _text()
68 {
69     _window.setKeyRepeatEnabled( true );
70     context._music->play( Music::Town_Peaceful_Place );
```

```
71      _text.setFont( context._fonts->get( Fonts::Connection_II ) );
72      _text.setFillColor( sf::Color::Black );
73      Logger::log( LoggerLevel::INFO, "Init levels" );
74      _levels = std::vector< std::string >();
75
76      if ( get_all_levels().empty() )
77      {
78          Logger::log( LoggerLevel::INFO, "No levels loaded" );
79          return;
80      }
81
82      for ( const boost::filesystem::path &path: get_all_levels() )
83      {
84          _levels.emplace_back( path.string() );
85      }
86
87      Logger::log( LoggerLevel::INFO, "Levels loaded" );
88      for ( const std::string &lvl: _levels )
89      {
90          Logger::log( LoggerLevel::INFO, lvl );
91      }
92      current_level = 0;
93      _text.setFont( context._fonts->get( Fonts::Connection_II ) );
94      _level = _levels.at( current_level );
95      Logger::log( LoggerLevel::DEBUG, "Level loaded: " + _level );
96      _world = new World( *context._window, model::Board( _level ), *context._fonts, *context._sounds );
97  }
98
99  /**
100  * Default destructor for the Game state
101  */
102 State_Game::~State_Game()
103 {
104     delete _world;
105 }
106
107 /**
108  * Realtime update of the visually available Game window
109  * @param dt The clock time
110  * @return always true
111  */
112 bool State_Game::update( sf::Time dt )
113 {
114     _world->update( dt );
115     return true;
116 }
117
118 /**
119  * Function to draw the visual components of the Game state
120  */
121 void State_Game::draw()
122 {
123     _world->draw();
124     if ( _world->is_board_completed() )
125     {
126         if ( _levels.size() <= current_level )
127         {
128             _text.setCharacterSize( 50 );
129             Utility::center_origin( _text );
130             _text.setString( L"All Levels Completed!" );
131             _text.setPosition( get_context()._window->getView().getSize() / 2.f );
132             get_context()._window->draw( _text );
133         }
134         else
135         {
136             _text.setCharacterSize( 50 );
137             Utility::center_origin( _text );
138             _text.setString( L"Level Completed\nPress ENTER to continue!" );
139             _text.setPosition( get_context()._window->getView().getSize() / 2.f );
140             get_context()._window->draw( _text );
141         }
142     }
143 }
144
145 /**
146  * Event handler meant for each interaction possible within the Game state
147  * @param event The event executed
148  * @return always true
149  */
150 bool State_Game::handle_event( const sf::Event &event )
151 {
152     if( event.type == sf::Event::KeyPressed )
153     {
154         if( _world->is_board_completed() )
```

```
155         {
156             if ( event.key.code == sf::Keyboard::Enter )
157             {
158                 next_level();
159             }
160         }
161         else
162         {
163             if( event.key.code == sf::Keyboard::Escape )
164             {
165                 request_stack_push( States::Pause );
166             }
167             else if( event.key.code == sf::Keyboard::R )
168             {
169                 reset_board();
170             }
171             else if( event.key.code == sf::Keyboard::S )
172             {
173                 next_level();
174             }
175             else if( event.key.code == sf::Keyboard::X )
176             {
177                 prev_level();
178             }
179             if( event.key.code == sf::Keyboard::Up )
180             {
181                 _world->move_up( true );
182             }
183             else if( event.key.code == sf::Keyboard::Down )
184             {
185                 _world->move_down( true );
186             }
187             else if( event.key.code == sf::Keyboard::Left )
188             {
189                 _world->move_left( true );
190             }
191             else if( event.key.code == sf::Keyboard::Right )
192             {
193                 _world->move_right( true );
194             }
195         }
196     }
197     else if( event.type == sf::Event::KeyReleased )
198     {
199         if( _world->is_board_completed() )
200         {
201         }
202         else
203         {
204             if( event.key.code == sf::Keyboard::Up )
205             {
206                 _world->move_up( false );
207             }
208             else if( event.key.code == sf::Keyboard::Down )
209             {
210                 _world->move_down( false );
211             }
212             else if( event.key.code == sf::Keyboard::Left )
213             {
214                 _world->move_left( false );
215             }
216             else if( event.key.code == sf::Keyboard::Right )
217             {
218                 _world->move_right( false );
219             }
220         }
221     }
222     return true;
223 }
224
225 /**
226  * Board resetting function meant to restart a game board
227  */
228 void State_Game::reset_board()
229 {
230     delete _world;
231     _world = new World( _window, model::Board( _level ), *get_context()._fonts, *get_context()._sounds );
232     reset_counter++;
233     _world->set_reset_counter( reset_counter );
234 }
235
236 /**
237  * Function meant to move onto the next level
238  */
```

```
239 void State_Game::next_level()
240 {
241     current_level += 1;
242     if ( _levels.size() <= current_level )
243     {
244         current_level = _levels.size() - 1;
245         return;
246     }
247     _level = _levels.at( current_level );
248     delete _world;
249     _world = new World( _window, model::Board( _level ), *get_context()._fonts, *get_context()._sounds );
250     reset_counter = 0;
251     _world->set_reset_counter( reset_counter );
252 }
253
254 /**
255  * Function meant to return to a previous level
256  */
257 void State_Game::prev_level()
258 {
259     current_level -= 1;
260     if ( current_level <= 0 )
261     {
262         current_level = 1;
263         return;
264     }
265     _level = _levels.at( current_level );
266     delete _world;
267     _world = new World( _window, model::Board( _level ), *get_context()._fonts, *get_context()._sounds );
268     reset_counter = 0;
269     _world->set_reset_counter( reset_counter );
270 }
```

9.9.6 State_Menu.hpp

```
1  #ifndef SOKOBAN_STATE_MENU_HPP
2  #define SOKOBAN_STATE_MENU_HPP
3
4  #include "State.hpp"
5  #include "../components/Container.hpp"
6
7  #include <SFML/Graphics/Text.hpp>
8  #include <SFML/Graphics/Sprite.hpp>
9
10 namespace sokoban
11 {
12     namespace ui
13     {
14         namespace gui
15         {
16             class State_Menu
17                     : public State
18             {
19             public:
20                 State_Menu( State_Stack &stack, Context context );
21                 void draw() override;
22                 bool update( sf::Time dt ) override;
23                 bool handle_event( const sf::Event &event ) override;
24             private:
25                 sf::Sprite _background_sprite; /** The background sprite */
26                 Container _container; /** Container storing the various components */
27             };
28         }
29     }
30 }
31
32 #endif //SOKOBAN_STATE_MENU_HPP
```

9.9.7 State_Menu.cpp

```
1  #include "State_Menu.hpp"
2
3  #include "../components/Button.hpp"
4  #include "../Utility.hpp"
5  #include "../../Resource_Holder.hpp"
6  #include "../Music_Player.hpp"
7
8  #include <SFML/Graphics/Text.hpp>
9  #include <SFML/Graphics/RenderWindow.hpp>
10
11 #include <cmath>
12
```

```cpp
using namespace sokoban::ui::gui;

/**
 * Default constructor for the Menu state
 * @param stack The stack containing the various states
 * @param context The context meant to access all the resources available
 */
State_Menu::State_Menu( State_Stack &stack, State::Context context )
        : State( stack, context )
        , _container()
{
    sf::Texture &texture = context._textures->get( Textures::Title_Screen );
    sf::Font &font = context._fonts->get( Fonts::Rampart_One );

    _background_sprite.setTexture( texture );
    Utility::center_origin( _background_sprite );
    _background_sprite.setPosition( context._window->getView().getSize() / 2.f );

    auto play_button = std::make_shared< Button >( context );
    play_button->set_text( "Play" );
    play_button->set_callback( [ this ] ()
    {
        request_stack_pop();
        request_stack_push( States::Game );
    });

    auto settings_button = std::make_shared< Button >( context );
    settings_button->set_text( "Settings" );
    settings_button->set_callback( [ this ] ()
    {
        request_stack_push( States::Settings );
    });

    auto exit_button = std::make_shared< Button >( context );
    exit_button->set_text( "Exit" );
    exit_button->set_callback( [ this ] ()
    {
        get_context()._window->close();
    });

    settings_button->setPosition( context._window->getView().getSize() / 2.f );
    settings_button->setOrigin( 100.f, 25.f );
    play_button->setPosition( settings_button->getPosition() - sf::Vector2f( 0, 100.f ) );
    play_button->setOrigin( 100.f, 25.f );
    exit_button->setPosition( settings_button->getPosition() + sf::Vector2f( 0, 100.f ) );
    exit_button->setOrigin( 100.f, 25.f );

    _container.pack( play_button );
    _container.pack( settings_button );
    _container.pack( exit_button );
}

/**
 * Visually draw the various components and entities with the state
 */
void State_Menu::draw()
{
    sf::RenderWindow &window = *get_context()._window;

    window.setView( window.getDefaultView() );
    window.draw( _background_sprite );
    window.draw( _container );
}

/**
 * Realtime update the various components
 * @param dt The clock time
 * @return always true
 */
bool State_Menu::update( sf::Time dt )
{
    return true;
}

/**
 * Event handler for the Menu container
 * @param event The event that has been called
 * @return always false
 */
bool State_Menu::handle_event( const sf::Event &event )
{
    _container.handle_event( event );
    return false;
}
```

9.9.8 State_Pause.hpp

```
#ifndef SOKOBAN_STATE_PAUSE_HPP
#define SOKOBAN_STATE_PAUSE_HPP

#include "State.hpp"
#include "../components/Container.hpp"

#include <SFML/Graphics/Sprite.hpp>
#include <SFML/Graphics/Text.hpp>

namespace sokoban
{
    namespace ui
    {
        namespace gui
        {
            class State_Pause
                    : public State
            {
            public:
                State_Pause( State_Stack &stack, Context context );
                void draw() override;
                bool update( sf::Time dt ) override;
                bool handle_event( const sf::Event &event ) override;
            private:
                sf::Sprite _background_sprite; /** The background sprite */
                sf::Text _paused_text; /** The text stating that the game is paused */
                Container _container; /** Container containing the various components */
            };
        }
    }
}

#endif //SOKOBAN_STATE_PAUSE_HPP
```

9.9.9 State_Pause.cpp

```
#include "State_Pause.hpp"

#include "../components/Button.hpp"
#include "../Utility.hpp"
#include "../../Resource_Holder.hpp"

#include <SFML/Graphics/Text.hpp>
#include <SFML/Graphics/RenderWindow.hpp>

#include <cmath>

using namespace sokoban::ui::gui;

/**
 * Default constructor for the Pause state
 * @param stack The stack containing the various states
 * @param context The context containing the various resources
 */
State_Pause::State_Pause( State_Stack &stack, State::Context context )
        : State( stack, context )
{
    sf::Texture &texture = context._textures->get( Textures::Title_Screen );
    sf::Font &font = context._fonts->get( Fonts::Rampart_One );
    sf::Vector2f view_size = context._window->getView().getSize();

    _background_sprite.setTexture( texture );
    Utility::center_origin( _background_sprite );
    _background_sprite.setPosition( view_size / 2.f );

    _paused_text.setFont( font );
    _paused_text.setString( "Paused" );
    _paused_text.setCharacterSize( 64.f );
    Utility::center_origin( _paused_text );
    _paused_text.setPosition( view_size.x / 2.f, view_size.y / 2.f - 200.f );

    auto resume_button = std::make_shared< Button >( context );
    resume_button->set_text( "Resume" );
    resume_button->set_callback( [ this ] ()
    {
        request_stack_pop();
    });

    auto settings_button = std::make_shared< Button >( context );
    settings_button->set_text( "Settings" );
```

```
45      settings_button->set_callback( [ this ] ()
46      {
47          request_stack_push( States::Settings );
48      });
49
50      auto back_to_main_menu_button = std::make_shared< Button >( context );
51      back_to_main_menu_button->set_text( "Main Menu" );
52      back_to_main_menu_button->set_callback( [ this ] ()
53      {
54          request_stack_pop();
55          request_stack_pop();
56          request_stack_push( States::Menu );
57      });
58
59      settings_button->setPosition( context._window->getView().getSize() / 2.f );
60      settings_button->setOrigin( 100.f, 25.f );
61      resume_button->setPosition( settings_button->getPosition() - sf::Vector2f( 0, 100.f ) );
62      resume_button->setOrigin( 100.f, 25.f );
63      back_to_main_menu_button->setPosition( settings_button->getPosition() + sf::Vector2f( 0, 100.f ) );
64      back_to_main_menu_button->setOrigin( 100.f, 25.f );
65
66      _container.pack( resume_button );
67      _container.pack( settings_button );
68      _container.pack( back_to_main_menu_button );
69  }
70
71  /**
72   * Visually display the various components that make up the state
73   */
74  void State_Pause::draw()
75  {
76      sf::RenderWindow &window = *get_context()._window;
77      window.setView( window.getDefaultView() );
78      window.draw( _paused_text );
79      window.draw( _container );
80  }
81
82  /**
83   * Realtime update the various components
84   * @param dt The clock time
85   * @return always true
86   */
87  bool State_Pause::update( sf::Time dt )
88  {
89      return true;
90  }
91
92  /**
93   * Event handler for the current pause state
94   * @param event The event that has been called
95   * @return always false
96   */
97  bool State_Pause::handle_event( const sf::Event &event )
98  {
99      _container.handle_event( event );
100     if( event.type == sf::Event::KeyPressed )
101     {
102         if( event.key.code == sf::Keyboard::Escape )
103         {
104             request_stack_pop();
105         }
106     }
107     return false;
108 }
```

9.9.10 State_Settings.hpp

```
1  #ifndef SOKOBAN_STATE_SETTINGS_HPP
2  #define SOKOBAN_STATE_SETTINGS_HPP
3
4  #include "State.hpp"
5  #include "../components/Container.hpp"
6  #include "../components/Button.hpp"
7
8  #include <SFML/Graphics/Sprite.hpp>
9  #include <SFML/Graphics/Text.hpp>
10
11 namespace sokoban
12 {
13     namespace ui
14     {
15         namespace gui
16         {
```

```
17              class State_Settings
18                      : public State
19              {
20              public:
21                  State_Settings( State_Stack &stack, Context context );
22                  void draw() override;
23                  bool update( sf::Time dt ) override;
24                  bool handle_event( const sf::Event &event ) override;
25              private:
26                  sf::Sprite _background_sprite; /** The background sprite */
27                  sf::Text _settings_text; /** The text stating Settings */
28                  Container _container; /** The container containing the various components */
29                  Button::Ptr _music_volume_button; /** The music volume button */
30                  Button::Ptr _sound_effect_volume_button; /** The sound effect volume button */
31                  void change_sound_effect_volume( float val );
32                  void change_music_volume( float val );
33              };
34          }
35      }
36  }
37
38  #endif //SOKOBAN_STATE_SETTINGS_HPP
```

9.9.11 State_Settings.cpp

```
1   #include "State_Settings.hpp"
2
3   #include "../components/Button.hpp"
4   #include "../Utility.hpp"
5   #include "../../Resource_Holder.hpp"
6   #include "../Music_Player.hpp"
7   #include "../Sound_Player.hpp"
8   #include "../components/Label.hpp"
9
10  #include <SFML/Graphics/Text.hpp>
11  #include <SFML/Graphics/RenderWindow.hpp>
12
13  #include <cmath>
14
15  using namespace sokoban::ui::gui;
16
17  namespace
18  {
19      float tmp_music_vol;
20      float tmp_sound_effect_vol;
21  }
22
23  /**
24   * Default constructor for the Settings state
25   * @param stack The stack containing the various states
26   * @param context The context containing the globally accessible resources
27   */
28  State_Settings::State_Settings( State_Stack &stack, Context context )
29          : State( stack, context )
30  {
31      sf::Texture &texture = context._textures->get( Textures::Title_Screen );
32      sf::Font &font = context._fonts->get( Fonts::Rampart_One );
33      sf::Vector2f view_size = context._window->getView().getSize();
34
35      _background_sprite.setTexture( texture );
36      Utility::center_origin( _background_sprite );
37      _background_sprite.setPosition( view_size / 2.f );
38
39      _music_volume_button = std::make_shared< Button >( context );
40      _music_volume_button->set_text( std::to_string( ( int ) get_context()._music->get_volume() ) );
41      _music_volume_button->set_toggle( true );
42
43      _sound_effect_volume_button = std::make_shared< Button >( context );
44      _sound_effect_volume_button->set_text( std::to_string( ( int ) get_context()._sounds->get_volume() ) );
45      _sound_effect_volume_button->set_toggle( true );
46
47      auto back_button = std::make_shared< Button >( context );
48      back_button->set_text( "Back" );
49      back_button->set_callback( [ this ] ()
50      {
51          request_stack_pop();
52      });
53
54      _sound_effect_volume_button->setPosition( context._window->getView().getSize() / 2.f );
55      _sound_effect_volume_button->setOrigin( 100.f, 25.f );
56
57      auto sound_effect_label = std::make_shared< Label >( "Sound Effect Volume", *get_context()._fonts, 24.f
        );
```

```
58      sound_effect_label->setPosition( _sound_effect_volume_button->getPosition() - sf::Vector2f( 0, 25.f ) );
59      sound_effect_label->setOrigin( 100.f, 25.f );
60
61      _music_volume_button->setPosition( _sound_effect_volume_button->getPosition() - sf::Vector2f( 0, 100.f )
          );
62      _music_volume_button->setOrigin( 100.f, 25.f );
63
64      auto music_label= std::make_shared< Label >( "Music Volume", *get_context()._fonts, 24.f );
65      music_label->setPosition( _music_volume_button->getPosition() - sf::Vector2f( 0, 25.f ) );
66      music_label->setOrigin( 100.f, 25.f );
67
68      back_button->setPosition( _sound_effect_volume_button->getPosition() + sf::Vector2f( 0, 100.f ) );
69      back_button->setOrigin( 100.f, 25.f );
70
71      _container.pack( _music_volume_button );
72      _container.pack( music_label );
73      _container.pack( _sound_effect_volume_button );
74      _container.pack( sound_effect_label );
75      _container.pack( back_button );
76
77      _settings_text.setFont( font );
78      _settings_text.setString( "Settings" );
79      _settings_text.setCharacterSize( 64.f );
80      Utility::center_origin( _settings_text );
81      _settings_text.setPosition( view_size.x / 2.f, 200.f );
82  }
83
84  /**
85   * Visually display the various components that make up the Settings state
86   */
87  void State_Settings::draw()
88  {
89      sf::RenderWindow &window = *get_context()._window;
90      window.setView( window.getDefaultView() );
91      window.draw( _settings_text );
92      window.draw( _container );
93  }
94
95  /**
96   * Realtime update the various components
97   * @param dt The clock time
98   * @return always true
99   */
100 bool State_Settings::update( sf::Time dt )
101 {
102     _music_volume_button->set_text( std::to_string( ( int ) get_context()._music->get_volume() ) );
103     _sound_effect_volume_button->set_text( std::to_string( ( int ) get_context()._sounds->get_volume() ) );
104     return true;
105 }
106
107 /**
108  * Event handler for the current state
109  * @param event The event that has been called
110  * @return always false
111  */
112 bool State_Settings::handle_event( const sf::Event &event )
113 {
114     bool is_setting_volume = false;
115
116     if( _sound_effect_volume_button->is_active() )
117     {
118         is_setting_volume = true;
119         if( event.type == sf::Event::KeyReleased )
120         {
121             if( event.key.code == sf::Keyboard::Enter )
122             {
123                 _sound_effect_volume_button->deactivate();
124             }
125             else if( event.key.code == sf::Keyboard::Left || event.key.code == sf::Keyboard::Down )
126             {
127                 change_sound_effect_volume( -5.f );
128             }
129             else if( event.key.code == sf::Keyboard::Right || event.key.code == sf::Keyboard::Up )
130             {
131                 change_sound_effect_volume( 5.f );
132             }
133         }
134     }
135     if( _music_volume_button->is_active() )
136     {
137         is_setting_volume = true;
138         if( event.type == sf::Event::KeyReleased )
139         {
140             if( event.key.code == sf::Keyboard::Enter )
```

```
141             {
142                 _music_volume_button->deactivate();
143             }
144             else if( event.key.code == sf::Keyboard::Left || event.key.code == sf::Keyboard::Down )
145             {
146                 change_music_volume( -5.f );
147             }
148             else if( event.key.code == sf::Keyboard::Right || event.key.code == sf::Keyboard::Up )
149             {
150                 change_music_volume( 5.f );
151             }
152         }
153     }
154     if( _sound_effect_volume_button->is_selected() && !is_setting_volume )
155     {
156         if( event.type == sf::Event::KeyReleased )
157         {
158             if( event.key.code == sf::Keyboard::M )
159             {
160                 float vol = get_context()._sounds->get_volume();
161                 if( vol > 0.f )
162                 {
163                     tmp_sound_effect_vol = get_context()._sounds->get_volume();
164                     vol = 0.f;
165                 }
166                 else
167                 {
168                     vol = tmp_sound_effect_vol;
169                 }
170                 get_context()._sounds->set_volume( vol );
171             }
172             else if( event.key.code == sf::Keyboard::Left )
173             {
174                 change_sound_effect_volume( -5.f );
175             }
176             else if( event.key.code == sf::Keyboard::Right )
177             {
178                 change_sound_effect_volume( 5.f );
179             }
180         }
181     }
182     if( _music_volume_button->is_selected() && !is_setting_volume )
183     {
184         if( event.type == sf::Event::KeyReleased )
185         {
186             if( event.key.code == sf::Keyboard::M )
187             {
188                 float vol = get_context()._music->get_volume();
189                 if( vol > 0.f )
190                 {
191                     tmp_music_vol = get_context()._music->get_volume();
192                     vol = 0.f;
193                 }
194                 else
195                 {
196                     vol = tmp_music_vol;
197                 }
198                 get_context()._music->set_volume( vol );
199             }
200             else if( event.key.code == sf::Keyboard::Left )
201             {
202                 change_music_volume( -5.f );
203             }
204             else if( event.key.code == sf::Keyboard::Right )
205             {
206                 change_music_volume( 5.f );
207             }
208         }
209     }
210
211     if( !is_setting_volume )
212     {
213         _container.handle_event( event );
214         if( event.type == sf::Event::KeyReleased )
215         {
216             if( event.key.code == sf::Keyboard::Escape )
217             {
218                 request_stack_pop();
219             }
220         }
221     }
222     return false;
223 }
224
```

```
225 /**
226  * Function to change the sound effect volume
227  * @param val The value that we wish to increase or decrease the volume with
228  */
229 void State_Settings::change_sound_effect_volume( float val )
230 {
231     float vol = get_context()._sounds->get_volume() + val;
232     if( vol < 0.f )
233     {
234         vol = 100.f;
235     }
236     if( vol > 100.f )
237     {
238         vol = 0.f;
239     }
240     get_context()._sounds->set_volume( vol );
241 }
242
243 /**
244  * Function to change the music volume
245  * @param val The value we wish to increase or decrease the volume with
246  */
247 void State_Settings::change_music_volume( float val )
248 {
249     float vol = get_context()._music->get_volume() + val;
250     if( vol < 0.f )
251     {
252         vol = 100.f;
253     }
254     else if( vol > 100.f )
255     {
256         vol = 0.f;
257     }
258     get_context()._music->set_volume( vol );
259 }
```

9.9.12 State_Stack.hpp

```
1  #ifndef SOKOBAN_STATE_STACK_HPP
2  #define SOKOBAN_STATE_STACK_HPP
3
4  #include "State.hpp"
5  #include "State_Identifiers.hpp"
6  #include "../Resource_Identifiers.hpp"
7
8  #include <SFML/System/Time.hpp>
9  #include <SFML/System/NonCopyable.hpp>
10
11 #include <map>
12 #include <vector>
13 #include <utility>
14 #include <functional>
15
16 namespace sf
17 {
18     class Event;
19     class RenderWindow;
20 }
21
22 namespace sokoban
23 {
24     namespace ui
25     {
26         namespace gui
27         {
28             class State_Stack
29             : private sf::NonCopyable
30                 {
31                 public:
32                     /**
33                      * The various actions possible with the stack
34                      */
35                     enum Action
36                     {
37                         Push, /** Push a state ahead */
38                         Pop, /** Pop an existing state */
39                         Clear /** Clear the entirety of the stack */
40                     };
41                     explicit State_Stack( State::Context context );
42                     template< typename T >
43                         void register_state( States::ID state_id );
44                     void update( sf::Time dt );
45                     void draw();
```

```
46                 void handle_event( const sf::Event &event );
47                 void push_state( States::ID state_id );
48                 void pop_state();
49                 void clear_states();
50                 bool is_empty() const;
51             private:
52                 State::Ptr create_state( States::ID state_id );
53                 void apply_pending_changes();
54                 struct Pending_Change
55                 {
56                     explicit Pending_Change( Action action, States::ID state_id = States::None );
57                     Action _action;
58                     States::ID _state_id;
59                 };
60                 std::vector< State::Ptr > _stack; /** The stack containing the various states */
61                 std::vector< Pending_Change > _pending_list; /** The handler meant to change the state based
       on an action */
62                 State::Context _context; /** The globally accessible resources */
63                 std::map< States::ID, std::function< State::Ptr() > > _factories; /** Mapper that defines a
      State per ID */
64             };
65
66             /**
67              * Dynamic initialisation of a state upon registration
68              * @tparam T The nature of the state
69              * @param state_id The ID that characterizes it
70              */
71             template< typename T >
72                 void State_Stack::register_state( States::ID state_id )
73                 {
74                     _factories[ state_id ] = [ this ] ()
75                     {
76                         return State::Ptr( new T( *this, _context ) );
77                     };
78                 }
79         }
80     }
81 }
82
83 #endif //SOKOBAN_STATE_STACK_HPP
```

9.9.13 State_Stack.cpp

```
1  #include "State_Stack.hpp"
2
3  #include <cassert>
4
5  using namespace sokoban::ui::gui;
6
7  /**
8   * Default constructor for the Stack containing the various states
9   * @param context Globally accessible context containing the various resources
10  */
11 State_Stack::State_Stack( State::Context context )
12 : _stack()
13 , _pending_list()
14 , _context( context )
15 , _factories()
16 {
17 }
18
19 /**
20  * Realtime update the various states within the stack
21  * @param dt The clock time
22  */
23 void State_Stack::update( sf::Time dt )
24 {
25     for( auto itr = _stack.rbegin(); itr != _stack.rend(); ++itr )
26     {
27         if( !( *itr )->update( dt ) )
28         {
29             break;
30         }
31     }
32     apply_pending_changes();
33 }
34
35 /**
36  * Visually display the various states within the stack
37  */
38 void State_Stack::draw()
39 {
40     for( State::Ptr &state : _stack )
```

```
41      {
42          state->draw();
43      }
44  }
45
46  /**
47   * Event handler for each state present within the stack
48   * @param event The event called
49   */
50  void State_Stack::handle_event( const sf::Event &event )
51  {
52      for( auto itr = _stack.rbegin(); itr != _stack.rend(); ++itr )
53      {
54          if( !( *itr )->handle_event( event ) )
55          {
56              break;
57          }
58      }
59      apply_pending_changes();
60  }
61
62  /**
63   * The state to push ahead of the stack
64   * @param state_id The ID that characterizes the state
65   */
66  void State_Stack::push_state( States::ID state_id )
67  {
68      _pending_list.emplace_back( Push, state_id );
69  }
70
71  /**
72   * Function that pops a state from within the stack
73   */
74  void State_Stack::pop_state()
75  {
76      _pending_list.emplace_back( Pop );
77  }
78
79  /**
80   * Function that clears out the entirety of the states within the stack
81   */
82  void State_Stack::clear_states()
83  {
84      _pending_list.emplace_back( Clear );
85  }
86
87  /**
88   * Verifier that checks whether the stack is empty or not
89   * @return Whether the stack is empty or not
90   */
91  bool State_Stack::is_empty() const
92  {
93      return _stack.empty();
94  }
95
96  /**
97   * Function that creates a state based on its ID
98   * @param state_id The ID that characterizes the state
99   * @return The State from its ID
100  */
101 State::Ptr State_Stack::create_state( States::ID state_id )
102 {
103     auto found = _factories.find( state_id );
104     assert( found != _factories.end() );
105     return found->second();
106 }
107
108 /**
109  * Function that handles the various interaction called within the game
110  */
111 void State_Stack::apply_pending_changes()
112 {
113     for( Pending_Change change : _pending_list )
114     {
115         switch ( change._action )
116         {
117             case Push:
118                 _stack.push_back( create_state( change._state_id ) );
119                 break;
120             case Pop:
121                 _stack.pop_back();
122                 break;
123             case Clear:
124                 _stack.clear();
```

```
125                 break;
126             }
127         }
128         _pending_list.clear();
129     }
130
131     /**
132      * Default constructor for a possible pending change
133      * @param action The action that has to be executed
134      * @param state_id The ID that characterizes the state that calls the action
135      */
136     State_Stack::Pending_Change::Pending_Change( State_Stack::Action action, States::ID state_id )
137     : _action( action )
138     , _state_id( state_id )
139     {
140     }
```

9.9.14 State_Title.hpp

```
1   #ifndef SOKOBAN_STATE_TITLE_HPP
2   #define SOKOBAN_STATE_TITLE_HPP
3
4   #include "State.hpp"
5
6   #include <SFML/Graphics/Text.hpp>
7   #include <SFML/Graphics/Sprite.hpp>
8
9   namespace sokoban
10  {
11      namespace ui
12      {
13          namespace gui
14          {
15              class State_Title
16                      : public State
17              {
18              public:
19                  State_Title( State_Stack &stack, Context context );
20                  void draw() override;
21                  bool update( sf::Time dt ) override;
22                  bool handle_event( const sf::Event &event ) override;
23              private:
24                  sf::Sprite _background_sprite; /** The background sprite */
25                  sf::Text _text; /** The flashing text */
26                  sf::Text _title_text; /** The text in japanese stating Sokoban */
27                  sf::Text _title_sub_text; /** The sub text translating Sokoban from Japanese */
28                  bool _show_text; /** Bool that states whether the text has to be hidden or not */
29                  sf::Time _text_effect_time; /** Timer for the text to blink */
30              };
31          }
32      }
33  }
34
35  #endif //SOKOBAN_STATE_TITLE_HPP
```

9.9.15 State_Title.cpp

```
1   #include "State_Title.hpp"
2
3   #include "../../../util/Logger.hpp"
4   #include "../Utility.hpp"
5   #include "../../Resource_Holder.hpp"
6
7   #include <SFML/Graphics/RenderWindow.hpp>
8
9   using namespace sokoban::ui::gui;
10  using namespace sokoban::util;
11
12  /**
13   * Default constructor for the Title state
14   * @param stack The stack containing the various states
15   * @param context The context containing the various resources
16   */
17  State_Title::State_Title( State_Stack &stack, State::Context context )
18          : State( stack, context )
19          , _text()
20          , _title_text()
21          , _show_text( true )
22          , _text_effect_time( sf::Time::Zero )
23  {
24      Logger::log( LoggerLevel::DEBUG, "Initializing Title Screen" );
25
```

```
26      Logger::log( LoggerLevel::DEBUG, "Loading Title Screen Texture" );
27      _background_sprite.setTexture( context._textures->get( Textures::Title_Screen ) );
28      Utility::center_origin( _background_sprite );
29      _background_sprite.setPosition( context._window->getView().getSize() / 2.f );
30
31      Logger::log( LoggerLevel::DEBUG, "Loading Title Screen Text" );
32
33      _title_text.setFont( context._fonts->get( Fonts::Kodomo_Rounded ) );
34      _title_text.setString( L"" );
35      Utility::center_origin( _title_text );
36      sf::Vector2f pos( context._window->getView().getSize() / 2.f );
37      _title_text.setPosition( pos.x / 1.5f, 10.f );
38      _title_text.setCharacterSize( 8 * 24 );
39      _title_text.setFillColor( sf::Color::Cyan );
40
41      _title_sub_text.setFont( context._fonts->get( Fonts::Kodomo_Rounded ) );
42      _title_sub_text.setString( "Sokoban" );
43      Utility::center_origin( _title_sub_text );
44      _title_sub_text.setPosition( pos.x / 1.2f, _title_text.getPosition().y + 200.f );
45      _title_sub_text.setCharacterSize( 4 * 24 );
46      _title_sub_text.setFillColor( sf::Color::Cyan );
47
48      _text.setFont( context._fonts->get( Fonts::Rampart_One ) );
49      _text.setString( "Press any key to start" );
50      Utility::center_origin( _text );
51      _text.setPosition( pos.x, pos.y + 150.f );
52      _text.setCharacterSize( 32 );
53  }
54
55  /**
56   * Visually display the various components within the state
57   */
58  void State_Title::draw()
59  {
60      sf::RenderWindow &window = *get_context()._window;
61      window.draw( _background_sprite );
62      window.draw( _title_text );
63      window.draw( _title_sub_text );
64      if( _show_text )
65      {
66          window.draw( _text );
67      }
68  }
69
70  /**
71   * Realtime update each component
72   * @param dt The clock time
73   * @return always true
74   */
75  bool State_Title::update( sf::Time dt )
76  {
77      _text_effect_time += dt;
78      if( _text_effect_time >= sf::seconds( 0.5f ) )
79      {
80          _show_text = !_show_text;
81          _text_effect_time = sf::Time::Zero;
82      }
83      return true;
84  }
85
86  /**
87   * Event handler for the current state
88   * @param event The event to manage
89   * @return always true
90   */
91  bool State_Title::handle_event( const sf::Event &event )
92  {
93      if( event.type == sf::Event::KeyReleased )
94      {
95          request_stack_pop();
96          request_stack_push( States::Menu );
97      }
98      return true;
99  }
```