SOKOBAN

Development report

Names: Hawraa Zein
Souheir Al-Khatib

Presented to: Dr. Antoun Yaacoub

Due date: July 5, 2018

Contents

Acknowledgments:

We express our sincere respect and gratitude to our subject teacher who has given his valuable support and suggestions from time to time to successfully develop this project.

Thanks to our classmates, or sincerely to our second family, who have been with us through difficult times. Thanks for all of us for being one hand for the sake of passing those 3 years with great success and for the sake of being true friends and one family.

Finally, big thanks to our parents who have always been encouraging us to overcome every obstacle we face and who have been raising us every time we felt we're down.

Abstract:

The project is about a game called Sokoban where the player pushes boxes or crates around in a warehouse, trying to get them to storage locations.

Introduction:

This report covers mainly the development process of Sokoban game. In order to be familiar with the strategy of playing the game, it is highly recommended that you check the user manual for the game.

So as mentioned in the abstract Sokoban is a type of transport puzzle, in which the player pushes boxes or crates around in a warehouse, trying to get them to storage locations.

Each level is completed when all of the boxes are at the storage locations. For further information about the rules of the game, please check the user manual.

Project summary:

Project title: Sokoban.

Project number: 20.

Project duration: 15 days.

Date of report: June 18, 2018.

Achievements:

- Learned new technology.

- Learned sharing ideas and gained management skills.

- Met deadlines consistently.

Overview:

The 3 main reasons behind the success of our project were: achievement, structure and power.

Need for achievement: being a specialist and advisor.

Being a technical specialist controlling your own area of responsibility.

Need for Structure: having support from systems and people

Structure - imposed systems

Having clear targets and objectives, a well structured environment, being able to establish effective methods, working within existing systems

Affiliation - support from colleagues

Having team support, being able to gain other people's views when working in less familiar areas

Need for power: being an organizer and implementer

Challenges:

Writing a javaFX project from scratch without having knowledge about its basics before. Throughout the next sections we will explain more about the basics of javaFx and how we developed this game using this language.

Project Progress:

1- Google:

Google was our best friend throughout the completion of this project. We googled the basics of javaFX, such as how to display pictures on the screen to how to add nodes to this screen, so that we succeeded to finish this project at time.

2- Getting images needed:

This process took a lot of time. Finding appropriate and user friendly images was one of the hardest tasks with respect to us.

3- Initialization of all variables needed:

- Initialization of the pane that will include everything that appears to the user.

- Initialization of the map size width and map size height (The height and width of the screen that will appear when you launch the game).

- Initialization of the area of the rectangle, which we will fill with images, in order to have equal sized images displayed.

- Initialization of 3 array lists of type 'Rectangle'. The first one stands for 'Blocks' and it contains the wall rectangles that will be used later on to limit the movements of Sokoban man. The second stands for 'winRects' that is used to know the place of each storage location. The last one stands for 'objRects' that will be used, in addition to 'winRects', in order to know if all boxes are at storage locations which means the completeness of the level. 'objRects' is also used in order to limit the movements of Sokoban man.

- Initialization of text variables that will be displayed on the screen.

- Initialization of all images needed to be displayed (such as wall, man and ground).

- Initialization of 'actions' Stack and 'rects' Stack. These stacks are useful for the undo function that will be discussed later on. The 'actions' Stack contains objects of type String while 'rects' stack contains objects of type 'Rectangle'. Whenever the man moves up, for instance, if the man while moving pushes with him a box, a 'UP-F' is pushed into 'actions' stack, where F stands for full, and the rectangle that contains the pushed box is pushed into 'rects' stack in order to catch its location later on when we want to undo. Otherwise, a 'UP-E' is pushed into the 'actions' stack, where E stands for empty, without pushing anything into 'rects' stack.

- Initialization of 'levelN' that will help us know the current level.

4- Implementation:

. 'undo' function:

```
public void undo() {
	if(!actions.isEmpty()) {
			String action=(String) actions.peek();
			actions.pop();
			String parts[] = action.split("-");
			String direction = parts[0];
			String condition = parts[1];
			if(condition.equals("E")) {
				switch(direction) {
				case "UP": man.setTranslateY(man.getTranslateY()+HW); break;
				case "DOWN": man.setTranslateY(man.getTranslateY()-HW); break;
				case "RIGHT": man.setTranslateX(man.getTranslateX()-HW); break;
				case "LEFT": man.setTranslateX(man.getTranslateX()+HW); break;
				}
			}

			else {
				if(condition.equals("F")){
				if(!rects.isEmpty()) {
				Rectangle object = (Rectangle) rects.peek();
				rects.pop();
				switch(direction) {
				case "UP": man.setTranslateY(man.getTranslateY()+HW);
object.setTranslateY(object.getTranslateY()+HW); break;
				case "DOWN": man.setTranslateY(man.getTranslateY()-HW);
object.setTranslateY(object.getTranslateY()-HW); break;
				case "RIGHT": man.setTranslateX(man.getTranslateX()-HW);
object.setTranslateX(object.getTranslateX()-HW); break;
				case "LEFT": man.setTranslateX(man.getTranslateX()+HW);
object.setTranslateX(object.getTranslateX()+HW); break;
				}}}
			}
		}
}
```

The undo function was implemented in order to help the user undo specific action(s) in case he / she pressed a keyboard key by mistake or if he / she made a specific action and they want to undo it instead of repeating the whole level. In the previous section we discussed when an object is pushed into each stack. Now we will discuss why we made 2 stacks and why we push sometimes just in one stack and other times in both stacks. First of all, the need of 2 stacks is obvious since each stack contains different type of objects. The 'actions' stack is needed to know how the user moves in order to know what to do when he presses undo. (For instance when the user presses the 'UP' key, an 'UP' action is pushed into the stack so that when he presses undo we move him back DOWN). The 'rects' stack is needed to know the exact object box pushed, if any, in order to catch it and change its position again when undo button is pressed. Now let's talk about how 'undo' function works.

First of all, we check if 'actions' stack is empty. If not, we catch the top of the stack using 'action' variable and we pop it. The 'action' variable is in the form X-Y so we split the 'action' variable into 2 parts in order to catch the direction (up, down, right, or left) and the condition (F which stands for full and E which stands for empty). If the condition was equal to 'E', we conclude that no box was pushed with the man so we do not touch the 'rects' stack for there is no need to that. We proceed and check the ‘direction’ variable. If the direction was equal to ‘UP’, we move the man down by increasing his y position by ‘HW’. If the direction was equal to ‘DOWN’, we move the man up by decreasing his y position by ‘HW’. If the direction was equal to ‘RIGHT’, we move the man left by decreasing his x position by ‘HW’. If the direction was equal to ‘LEFT’, we move the man right by increasing his x position by ‘HW’. Now, if the condition was equal to ‘F’, we conclude that a box was pushed with the man during his last movement so we need to access the ‘rects’ stack in order to catch the pushed box. We catch the top of ‘rects’ stack and we pop, then we repeat the above conditions but this time when we change the position of the man, we have also to change the position of the box.

. ‘addInfo’ function:

```
public void addInfo() {
        text2.setTranslateX(430);
        text2.setTranslateY(280);
        text2.setText("Current Moves: " + moves);
        text2.setFont(Font.font("Comic Sans", 20));
        text2.setFill(Color.PERU);
        root.getChildren().add(text2);
       Text text1 = new Text();
       switch(levelN) {
       case 1:
            text1.setTranslateX(445);
            text1.setTranslateY(250);
            text1.setText("Best Moves: 31");
            text1.setFont(Font.font("Comic Sans", 20));
            text1.setFill(Color.PERU);
            root.getChildren().add(text1);
            break;
       case 2:
             text1.setTranslateX(445);
            text1.setTranslateY(250);
            text1.setText("Best Moves: 51");
            text1.setFont(Font.font("Comic Sans", 20));
            text1.setFill(Color.PERU);
            root.getChildren().add(text1);
            break;
       case 3:
             text1.setTranslateX(445);
            text1.setTranslateY(250);
            text1.setText("Best Moves: 76");
            text1.setFont(Font.font("Comic Sans", 20));
            text1.setFill(Color.PERU);
            root.getChildren().add(text1);
            break;
       case 4:
             text1.setTranslateX(445);
            text1.setTranslateY(250);
            text1.setText("Best Moves: 77");
            text1.setFont(Font.font("Comic Sans", 20));
            text1.setFill(Color.PERU);
            root.getChildren().add(text1);
            break;
      case 5:
             text1.setTranslateX(445);
            text1.setTranslateY(250);
            text1.setText("Best Moves: 81");
            text1.setFont(Font.font("Comic Sans", 20));
            text1.setFill(Color.PERU);
            root.getChildren().add(text1);
            break;
      case 6:
             text1.setTranslateX(445);
            text1.setTranslateY(250);
            text1.setText("Best Moves: 83");
            text1.setFont(Font.font("Comic Sans", 20));
            text1.setFill(Color.PERU);
            root.getChildren().add(text1);
            break;
      case 7:
             text1.setTranslateX(445);
            text1.setTranslateY(250);
            text1.setText("Best Moves: 87");
            text1.setFont(Font.font("Comic Sans", 20));
            text1.setFill(Color.PERU);
            root.getChildren().add(text1);
            break;
      case 8:
             text1.setTranslateX(445);
            text1.setTranslateY(250);
            text1.setText("Best Moves: 90");
            text1.setFont(Font.font("Comic Sans", 20));
            text1.setFill(Color.PERU);
            root.getChildren().add(text1);
            break;
      case 9:
             text1.setTranslateX(445);
            text1.setTranslateY(250);
            text1.setText("Best Moves: 92");
            text1.setFont(Font.font("Comic Sans", 20));
            text1.setFill(Color.PERU);
            root.getChildren().add(text1);
            break;
      case 10:
             text1.setTranslateX(445);
            text1.setTranslateY(250);
            text1.setText("Best Moves: 94");
            text1.setFont(Font.font("Comic Sans", 20));
            text1.setFill(Color.PERU);
            root.getChildren().add(text1);
            break;

      }
   }
```

This function was implemented in order to add information for each level. The information displayed by this function are ‘Current moves’ and ‘Best moves’. These information are saved as text which is added to the pane. We switch on levelN in order to know in which level the user is in order to display the best moves and current moves according to which level the user is playing.

. ‘addRightMenu’ function:

```
public void addRightMenu() {

       addInfo();
        text.setTranslateX(455);
        text.setTranslateY(40);
        text.setText("Level " + (levelN));
        text.setFont(Font.font("Comic Sans", 30));
        text.setFill(Color.PERU);
        root.getChildren().add(text);

        Button retry = new Button();
        retry.setTranslateX(478);
        retry.setTranslateY(70);
        retry.setText("Retry");
        retry.setOnAction(e -> {
            root.getChildren().clear();
            Blocks.clear();
            objRects.clear();
            winRects.clear();
            changeLevel();
        });
        root.getChildren().add(retry);

        Button menu = new Button();
        menu.setTranslateX(478);
        menu.setTranslateY(110);
        menu.setText("Menu");
        menu.setOnAction(e -> {
             levelN=0;
            root.getChildren().clear();
            Blocks.clear();
            objRects.clear();
            winRects.clear();
            changeLevel();
        });
        root.getChildren().add(menu);

        Button exit = new Button();
        exit.setTranslateX(483);
        exit.setTranslateY(150);
        exit.setText("Exit");
        exit.setOnAction(e -> {
            System.exit(0);
        });
        root.getChildren().add(exit);

        Button undo = new Button();
        undo.setTranslateX(478);
        undo.setTranslateY(190);
        undo.setText("Undo");
        undo.setOnAction(e -> {
            undo();
        });
        root.getChildren().add(undo);
    }
```

This function is used to display a right menu whenever the user selects a level. This right menu includes in which level the user is, 'Exit' button, 'Undo' button, 'Retry' button, 'Menu' button, best moves and current moves. When the user presses the 'Undo' button, for instance, the function 'undo' is called.

. 'changeLevel' function:

```
public void changeLevel() {
       moves=0;
        switch (levelN) {
            case 0:
                Menu();
                break;
            case 1:
                level1();
                break;
            case 2:
                level2();
                break;
            case 3:
                level3();
                break;
            case 4:
                level4();
                break;
            case 5:
                level5();
                break;
            case 6:
                level6();
                break;
            case 7:
                level7();
                break;
            case 8:
                level8();
                break;
            case 9:
                level9();
                break;
            case 10:
                level10();
                break;
            case 11:
             Menu();
                break;

        }
    }
```

This function was implemented in order to change the level according to the value of levelN variable. If the user chooses level 2, levelN is equal to 2, thus level2() function is called in order to display level 2 for the user. Whenever the user wins a level, levelN increases by default, and by this when we complete level 2, for example, level 3 will be displayed after that.

. 'Menu' function:

```
public void Menu() {
        root.getChildren().clear();
        Blocks.clear();
        objRects.clear();
        winRects.clear();
        actions.clear();
        rects.clear();
        Rectangle b1;
       for (int i = 0; i < 600; i += HW) {
             for(int j=0 ; j<600 ;j+=HW){
            b1 = new Rectangle(HW, HW, new ImagePattern(ground));
            b1.setTranslateX(i);
            b1.setTranslateY(j);
            root.getChildren().add(b1);
             }
       }

       ImageView l1 = new ImageView(level1);
        l1.setFitHeight(50);
        l1.setFitWidth(50);
        Button lev1 = new Button("Level 1");
        lev1.setGraphic(l1);
        lev1.setTranslateX(50);
        lev1.setTranslateY(30);
        lev1.setOnAction(e -> {
            root.getChildren().clear();
            Blocks.clear();
            objRects.clear();
```

```
    winRects.clear();
    levelN = 1;
    changeLevel();
});
root.getChildren().add(lev1);

ImageView l2 = new ImageView(level2);
l2.setFitHeight(50);
l2.setFitWidth(50);
Button lev2 = new Button("Level 2");
lev2.setGraphic(l2);
lev2.setTranslateX(50);
lev2.setTranslateY(95);
lev2.setOnAction(e -> {
    root.getChildren().clear();
    Blocks.clear();
    objRects.clear();
    winRects.clear();
    levelN = 2;
    changeLevel();
});
root.getChildren().add(lev2);

ImageView l3 = new ImageView(level3);
l3.setFitHeight(50);
l3.setFitWidth(50);
Button lev3 = new Button("Level 3");
lev3.setGraphic(l3);
lev3.setTranslateX(50);
lev3.setTranslateY(160);
lev3.setOnAction(e -> {
    root.getChildren().clear();
    Blocks.clear();
    objRects.clear();
    winRects.clear();
    levelN = 3;
    changeLevel();
});
root.getChildren().add(lev3);

ImageView l4 = new ImageView(level4);
l4.setFitHeight(50);
l4.setFitWidth(50);
Button lev4 = new Button("Level 4");
lev4.setGraphic(l4);
lev4.setTranslateX(50);
lev4.setTranslateY(225);
lev4.setOnAction(e -> {
    root.getChildren().clear();
    Blocks.clear();
    objRects.clear();
    winRects.clear();
    levelN = 4;
    changeLevel();
});
```

```
root.getChildren().add(lev4);

ImageView l5 = new ImageView(level5);
l5.setFitHeight(50);
l5.setFitWidth(50);
Button lev5 = new Button("Level 5");
lev5.setGraphic(l5);
lev5.setTranslateX(50);
lev5.setTranslateY(290);
lev5.setOnAction(e -> {
    root.getChildren().clear();
    Blocks.clear();
    objRects.clear();
    winRects.clear();
    levelN = 5;
    changeLevel();
});
root.getChildren().add(lev5);

ImageView l6 = new ImageView(level6);
l6.setFitHeight(50);
l6.setFitWidth(50);
Button lev6 = new Button("Level 6");
lev6.setGraphic(l6);
lev6.setTranslateX(435);
lev6.setTranslateY(30);
lev6.setOnAction(e -> {
    root.getChildren().clear();
    Blocks.clear();
    objRects.clear();
    winRects.clear();
    levelN = 6;
    changeLevel();
});
root.getChildren().add(lev6);

ImageView l7 = new ImageView(level7);
l7.setFitHeight(50);
l7.setFitWidth(50);
Button lev7 = new Button("Level 7");
lev7.setGraphic(l7);
lev7.setTranslateX(435);
lev7.setTranslateY(95);
lev7.setOnAction(e -> {
    root.getChildren().clear();
    Blocks.clear();
    objRects.clear();
    winRects.clear();
    levelN = 7;
    changeLevel();
});
root.getChildren().add(lev7);

ImageView l8 = new ImageView(level8);
l8.setFitHeight(50);
```

```
l8.setFitWidth(50);
Button lev8 = new Button("Level 8");
lev8.setGraphic(l8);
lev8.setTranslateX(435);
lev8.setTranslateY(160);
lev8.setOnAction(e -> {
    root.getChildren().clear();
    Blocks.clear();
    objRects.clear();
    winRects.clear();
    levelN = 8;
    changeLevel();
});
root.getChildren().add(lev8);

ImageView l9 = new ImageView(level9);
l9.setFitHeight(50);
l9.setFitWidth(50);
Button lev9 = new Button("Level 9");
lev9.setGraphic(l9);
lev9.setTranslateX(435);
lev9.setTranslateY(225);
lev9.setOnAction(e -> {
    root.getChildren().clear();
    Blocks.clear();
    objRects.clear();
    winRects.clear();
    levelN = 9;
    changeLevel();
});
root.getChildren().add(lev9);

ImageView l10 = new ImageView(level10);
l10.setFitHeight(50);
l10.setFitWidth(45);
Button lev10 = new Button("level 10");
lev10.setGraphic(l10);
lev10.setTranslateX(435);
lev10.setTranslateY(290);
lev10.setOnAction(e -> {
    root.getChildren().clear();
    Blocks.clear();
    objRects.clear();
    winRects.clear();
    levelN = 10;
    changeLevel();
});
root.getChildren().add(lev10);

Button exit = new Button();
exit.setTranslateX(273);
exit.setTranslateY(210);
exit.setText("Exit");
exit.setOnAction(e -> {
    System.exit(0);
```

```
        });
        root.getChildren().add(exit);

        Button btn2 = new Button();
        btn2.setTranslateX(255);
        btn2.setTranslateY(170);
        btn2.setText("New Game");
        btn2.setOnAction(e -> {
            root.getChildren().clear();
            Blocks.clear();
            objRects.clear();
            winRects.clear();
            levelN = 1;
            changeLevel();
        });
        root.getChildren().add(btn2);

        text.setTranslateX(200);
        text.setTranslateY(90);
        text.setText("SOKOBAN");
        text.setFont(Font.font("Comic Sans", 40));
        text.setFill(Color.PERU);
        root.getChildren().add(text);
    }
```

This function was implemented so that when the user first launches the game, this function is called in order to add children to the pane so that the screen doesn't appear empty. (In order to add a children to the pane, we need to determine its x and its y and to fill it with appropriate objects.) Thus, when the user launches the game, the menu will appear, leaving the choice to the user to select a level from 1 to 10. 'Exit' button is used to exit the game. If the user chooses 'New Game', level 1 will start.

. 'level1' function:

```
public void level1() {
       Rectangle b1;
       for (int i = 0; i < 600; i += HW) {
              for(int j=0 ; j<600 ;j+=HW){
             b1 = new Rectangle(HW, HW, new ImagePattern(ground));
             b1.setTranslateX(i);
             b1.setTranslateY(j);
             root.getChildren().add(b1);
              }
       }

        Rectangle winRec;
        winRec = new Rectangle(HW, HW, new ImagePattern(place));
        winRec.setTranslateY(100);
        winRec.setTranslateX(150);
```

```
winRects.add(winRec);
root.getChildren().add(winRec);
winRec = new Rectangle(HW, HW, new ImagePattern(place));
winRec.setTranslateY(100);
winRec.setTranslateX(200);
winRects.add(winRec);
root.getChildren().add(winRec);
winRec = new Rectangle(HW, HW, new ImagePattern(place));
winRec.setTranslateY(100);
winRec.setTranslateX(300);
winRects.add(winRec);
root.getChildren().add(winRec);
winRec = new Rectangle(HW, HW, new ImagePattern(place));
winRec.setTranslateY(150);
winRec.setTranslateX(150);
winRects.add(winRec);
root.getChildren().add(winRec);

Rectangle objRec;
objRec = new Rectangle(HW, HW, new ImagePattern(box));
objRec.setTranslateY(100);
objRec.setTranslateX(150);
objRects.add(objRec);
root.getChildren().add(objRec);
objRec = new Rectangle(HW, HW, new ImagePattern(box));
objRec.setTranslateY(100);
objRec.setTranslateX(250);
objRects.add(objRec);
root.getChildren().add(objRec);
objRec = new Rectangle(HW, HW, new ImagePattern(box));
objRec.setTranslateY(150);
objRec.setTranslateX(250);
objRects.add(objRec);
root.getChildren().add(objRec);
objRec = new Rectangle(HW, HW, new ImagePattern(box));
objRec.setTranslateY(250);
objRec.setTranslateX(150);
objRects.add(objRec);
root.getChildren().add(objRec);

for (int i = 0; i < 400; i += HW) {
    b1 = new Rectangle(HW, HW, new ImagePattern(wall));
    b1.setTranslateX(i);
    b1.setTranslateY(0);
    Blocks.add(b1);
    root.getChildren().add(b1);
}


b1 = new Rectangle(HW, HW, new ImagePattern(wall));
b1.setTranslateX(0);
b1.setTranslateY(50);
Blocks.add(b1);
root.getChildren().add(b1);
```

```
for (int i = 150; i < 400; i += HW) {
    b1 = new Rectangle(HW, HW, new ImagePattern(wall));
    b1.setTranslateX(i);
    b1.setTranslateY(50);
    Blocks.add(b1);
    root.getChildren().add(b1);
}

b1 = new Rectangle(HW, HW, new ImagePattern(wall));
b1.setTranslateX(0);
b1.setTranslateY(100);
Blocks.add(b1);
root.getChildren().add(b1);

b1 = new Rectangle(HW, HW, new ImagePattern(wall));
b1.setTranslateX(350);
b1.setTranslateY(100);
Blocks.add(b1);
root.getChildren().add(b1);

b1 = new Rectangle(HW, HW, new ImagePattern(wall));
b1.setTranslateX(0);
b1.setTranslateY(150);
Blocks.add(b1);
root.getChildren().add(b1);

b1 = new Rectangle(HW, HW, new ImagePattern(wall));
b1.setTranslateX(350);
b1.setTranslateY(150);
Blocks.add(b1);
root.getChildren().add(b1);

for (int i = 0; i < 150; i += HW) {
    b1 = new Rectangle(HW, HW, new ImagePattern(wall));
    b1.setTranslateX(i);
    b1.setTranslateY(200);
    Blocks.add(b1);
    root.getChildren().add(b1);
}

for (int i = 250; i < 400; i += HW) {
    b1 = new Rectangle(HW, HW, new ImagePattern(wall));
    b1.setTranslateX(i);
    b1.setTranslateY(200);
    Blocks.add(b1);
    root.getChildren().add(b1);
}

for (int i = 0; i < 150; i += HW) {
    b1 = new Rectangle(HW, HW, new ImagePattern(wall));
    b1.setTranslateX(i);
    b1.setTranslateY(250);
    Blocks.add(b1);
    root.getChildren().add(b1);
}
```

```
        for (int i = 250; i < 400; i += HW) {
            b1 = new Rectangle(HW, HW, new ImagePattern(wall));
            b1.setTranslateX(i);
            b1.setTranslateY(250);
            Blocks.add(b1);
            root.getChildren().add(b1);
        }

        for (int i = 0; i < 150; i += HW) {
            b1 = new Rectangle(HW, HW, new ImagePattern(wall));
            b1.setTranslateX(i);
            b1.setTranslateY(300);
            Blocks.add(b1);
            root.getChildren().add(b1);
        }

        for (int i = 250; i < 400; i += HW) {
            b1 = new Rectangle(HW, HW, new ImagePattern(wall));
            b1.setTranslateX(i);
            b1.setTranslateY(300);
            Blocks.add(b1);
            root.getChildren().add(b1);
        }

        for (int i = 0; i < 400; i += HW) {
            b1 = new Rectangle(HW, HW, new ImagePattern(wall));
            b1.setTranslateX(i);
            b1.setTranslateY(350);
            Blocks.add(b1);
            root.getChildren().add(b1);
        }

        man = new Rectangle(HW, HW, new ImagePattern(imgman));
        man.setTranslateY(100);
        man.setTranslateX(300);
        root.getChildren().add(man);

        addRightMenu();
    }
```

This function was implemented so that it is called to add children to the pane when level 1 is selected from the Menu. We initialize an object 'Rectangle' so that we can add it to the pane later on. We fill all of the background by the image of ground and add them to the pane by looping from 0 to 599 and adding HW each time (The size of rectangle is HW). Then, we choose the positions of storage locations and we add them to the pane and to the 'winRects' so that we know their location later on if we want to check if boxes are at them. Then, we choose the positions of boxes and we add them to the pane and to the 'objRects' so that we can check if the man can move or to check if they are at the storage locations. After that we choose the positions of walls and we add them to the pane and to 'Blocks' so that we can limit the movements of the man. (For example, if the man wants to move up but above him there was a wall, then he can't move. This is checked by looping through the array list 'Blocks' and checking if the position of the wall is above the position of the man). Then, we choose the position of the man and determine his x and y so that we can move him when a keyboard key ('UP','DOWN','RIGHT','LEFT') is pressed. Finally, addRightMenu function is called in order to display a right menu in the current level.

level2(), level3(), level4(), level5(), level6(), level7(), level8(), level9(), and level10() share the same concept with level1() function. The only difference is in the positions of the objects.

. 'objOnWinRec' function:

```
public void objOnWinRec() {
        Rectangle objRec, winRec;
        int check = 0;
        for (int i = 0; i < objRects.size(); i++) {
            objRec = objRects.get(i);
            for (int j = 0; j < winRects.size(); j++) {
                winRec = winRects.get(j);
                if (objRec.getTranslateX() == winRec.getTranslateX() &&
objRec.getTranslateY() == winRec.getTranslateY()) {
                    check++;
                }
            }
        }
        if (check == objRects.size()) {
            levelN++;
            root.getChildren().clear();
            Blocks.clear();
            objRects.clear();
            winRects.clear();
            actions.clear();
            rects.clear();
            changeLevel();
        }
    }
```

This function was implemented in order to check if a box is at a storage location. Each level has a specific number of boxes which is equal to the number of storage locations. We use 'check' variable as a flag. Whenever we find a box at a storage location, we increment 'check' by one. If at last, we found that 'check' is equal to objRects.size(), we conclude that all boxes are at the storage locations, so we clear the children of the current level so that we can replace them with the children of the other level and we increment levelN by 1 so that the following level is displayed.

. ‘moveDown’ function:

```
public void moveDown() {
        int check = 0, check1 = 0;
        Rectangle objRec = null;
        for (int i = 0; i < objRects.size(); i++) {
            if (man.getTranslateX() == objRects.get(i).getTranslateX() &&
(man.getTranslateY() + HW) == objRects.get(i).getTranslateY()) {
                objRec = objRects.get(i);
                check1 = 1;
            }
        }

        if (check1 == 1) {
            for (int i = 0; i < objRects.size(); i++) {
                if (objRec.getTranslateX() == objRects.get(i).getTranslateX() &&
(objRec.getTranslateY() + HW) == objRects.get(i).getTranslateY()) {
                    check1 = 2;
                }
            }
        }

        if (check1 == 1) {
            if (man.getTranslateX() == objRec.getTranslateX() &&
objRec.getTranslateY() != mapsizeH - HW && man.getTranslateY() ==
(objRec.getTranslateY() - HW)) {
                for (int i = 0; i < Blocks.size(); i++) {
                    if (objRec.getTranslateY() + HW == Blocks.get(i).getTranslateY()
&& objRec.getTranslateX() == Blocks.get(i).getTranslateX()) {
                        check = 1;
                    }
                }

                if (check == 0) {
                    man.setTranslateY(man.getTranslateY() + HW);
                    objRec.setTranslateY(man.getTranslateY() + HW);
                    actions.push("DOWN-F");
                    rects.push(objRec);
                    moves++;
                }
            }

            else if (man.getTranslateX() == objRec.getTranslateX() &&
man.getTranslateY() != (objRec.getTranslateY() - HW)) {
                man.setTranslateY(man.getTranslateY() + HW);
                moves++;
                actions.push("DOWN-E");
            }

            else if (man.getTranslateX() != objRec.getTranslateX()) {
                man.setTranslateY(man.getTranslateY() + HW);
                moves++;
                actions.push("DOWN-E");
            }
        }

        else if (check1 == 0) {
            if (man.getTranslateY() != mapsizeH - HW) {
                man.setTranslateY(man.getTranslateY() + HW);
                moves++;
                actions.push("DOWN-E");
            }
        }
    }
```

This function was implemented in order to move the man or the box down if conditions are appropriate. We initialize 'check' and 'check1' variables as flags and we initialize an object Rectangle. We loop through the 'objRects' array list and each time we check the x and y of both: the man and objRect.get(i). If the man and the object share the same x and the object has y + HW and the man has y, we can conclude that below the man there is a box and we change check1 from 0 to 1 as a flag to know that there is a box under the man. Now, if there is a box under the man, we need to check if there is a box or a wall under the box that is under the man so that we can limit movements. We check that by the same manner mentioned above (checking x and y). If there is a box under the box, check1 will be equal to. If there is a wall under the box, check will be equal to 1. Now if check1=1 and check=0, then there is a man and a box under him, so we move down the man and the box by increasing their y's by HW and we push 'DOWN-F' and the current box to the stacks. If there was no box under the man, we only change the position of the man and push 'DOWN-E' to the stack. As the height of the map is mapsizeH then we need to check that if the man moves down he will not exceed that height by (man.getTranslateY() != mapsizeH – HW).

moveUp(), moveLeft() and moveRight() share the same manner with moveDown() function, obviously with different conditions. (check the full code for further information).

. 'manBlockColiDown' function:

```
public int manBlockColiDown() {
        int i = 0;
        for (i = 0; i < Blocks.size(); i++) {
            if (man.getTranslateY() == (Blocks.get(i).getTranslateY() - HW) &&
man.getTranslateX() == Blocks.get(i).getTranslateX()) {
                return 1;
            }
        }
        return 0;
    }
```

As we observed, moveDown() function is somehow big so in order to avoid its complexity and for the game to be optimized as much as we can, this function was implemented so that if there was a wall below the man we conclude that he can't move down and thus there is no need to go through moveDown() function.

manBlockColiUp(), manBlockColiLeft(), and manBlockColiRight() share the same manner with manBlockColiDown(), obviously with difference in conditions. (For further information please check the full code).

. 'changeManImage' function:

```
public void changeManImage(char direction){
        switch(direction){
            case 'w':
                man.setFill(new ImagePattern(imgman));
                break;
            case 'd':
                 man.setFill(new ImagePattern(right));
                break;
            case 's':
                 man.setFill(new ImagePattern(down));
                break;
            case 'a':
                 man.setFill(new ImagePattern(left));
                break;
        }
    }
```

This function was implemented so that we can change the man's image when he moves. If he moves right, we display an image of him moving right. If he moves down, we display an image of him moving down. If he moves up, we display an image of him moving up. If he moves left, we display an image of him moving left. This is achieved by changing the fill of man rectangle. The purpose of that is to make the user feel as if the man is really moving and have more fun while playing our game.

. 'start' function:

```
public void start(Stage stagee) {
        stage = stagee;
        stage.setScene(new Scene(createContent()));
        stage.getScene().setOnKeyPressed(event -> {
            if (levelN > 0 && levelN <= 10) {
                switch (event.getCode()) {
                    case UP: {
                        if (man.getTranslateY() != 0 && manBlockColiUP() != 1) {
                            moveUp();
                            text2.setText("Current Moves: " + moves);
                            objOnWinRec();
                            changeManImage('w');
                        }
                    }
                    break;
                    case DOWN:
                        if (man.getTranslateY() != mapsizeH - HW &&
manBlockColiDown() != 1) {
                            moveDown();

                            text2.setText("Current Moves: " + moves);
                            objOnWinRec();
                            changeManImage('s');
                        }
                        break;
                    case LEFT:
                        if (man.getTranslateX() != 0 && manBlockColiLeft() != 1) {
                            moveLeft();

                            text2.setText("Current Moves: " + moves);
                            objOnWinRec();
                            changeManImage('a');
                        }

                        break;
                    case RIGHT:
                        if (man.getTranslateX() != mapsizeW - HW &&
manBlockColiRight() != 1) {
                            moveRight();

                            text2.setText("Current Moves: " + moves);
                            objOnWinRec();
                            changeManImage('d');
                        }
                        break;
                    default:
                        break;
                }
            }
        });
        stage.setTitle("SOKOBAN");
        stage.show();
    }
}
```

The stage is what appears when you launch the game. So we fill the scene of this stage by the pane previously filled with children. Whenever a keyboard key is pressed, we check if levelN is between 1 and 10. If the user presses ‘UP’, we check if the user is at the top of the stage or if there is a wall above him. If not, we call moveUp function so that the man moves up and we check if all boxes are at storage locations and we change the image of the man. Same manner if he presses ‘DOWN’, ‘LEFT’, or ‘RIGHT’.

Finally, we call stage.show() so that the stage appears when launching the game.

. ‘createContent’ function:

```
public Parent createContent() {
        root = new Pane();
        root.setPrefSize(mapsizeW + 200, mapsizeH);
        Menu();
        return root;
    }
```

This function is used to create the final content of the pane that will be displayed on stage.

. ‘main’ function:

```
public static void main(String[] args) {
        launch(args);
    }
```

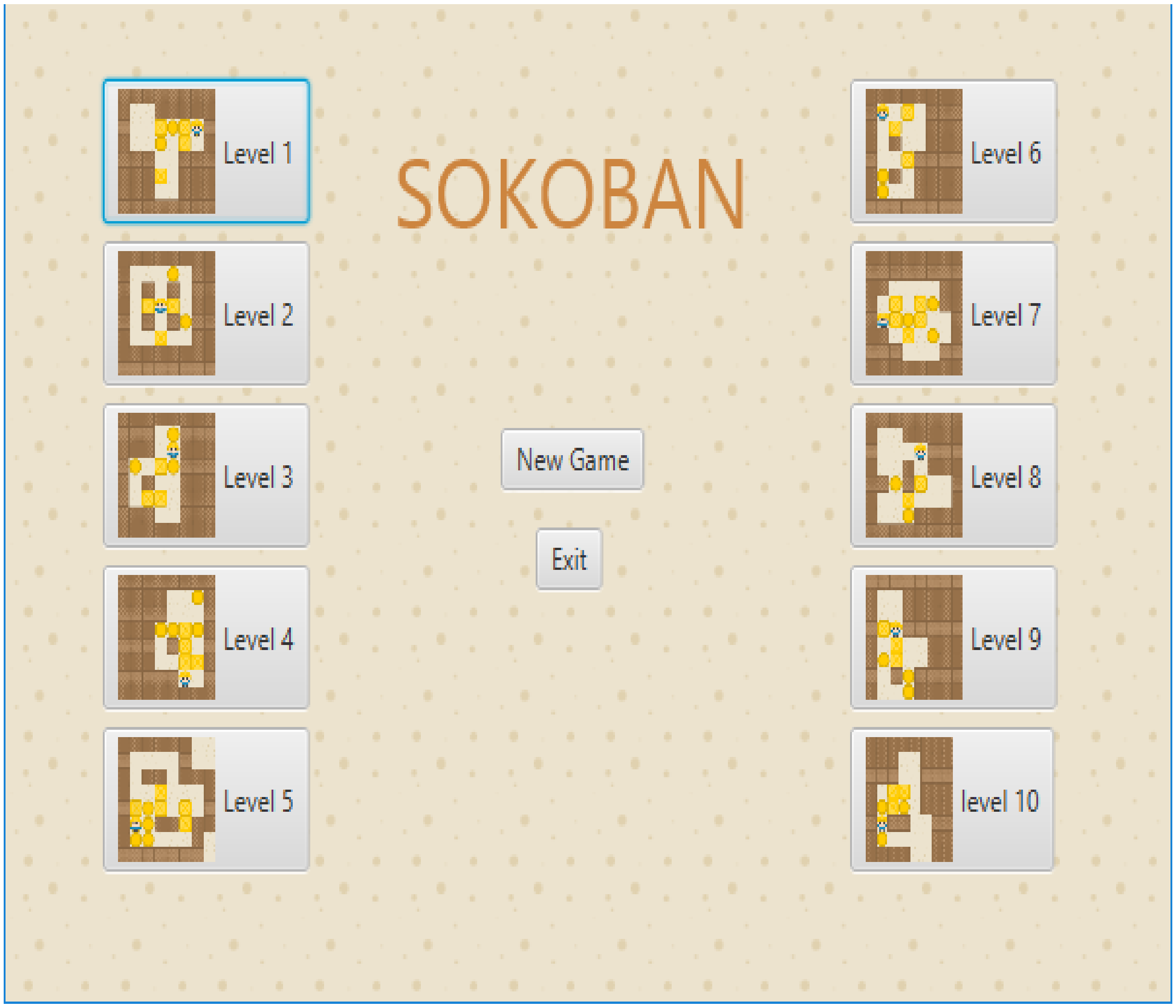
Without this function, the game will not launch.

5- Creating executable jar file:

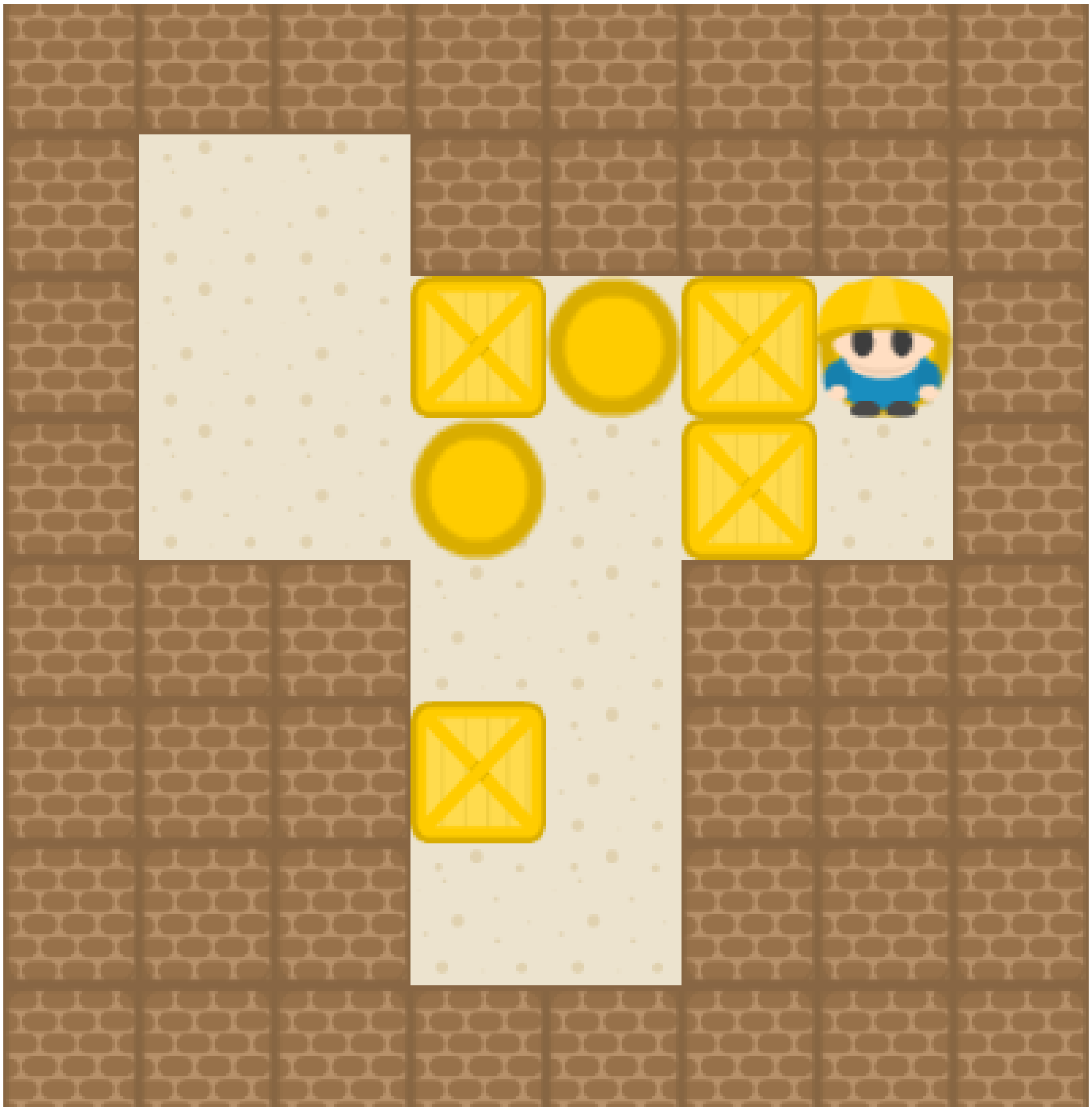
We exported the project as executable jar file.

Visualization:

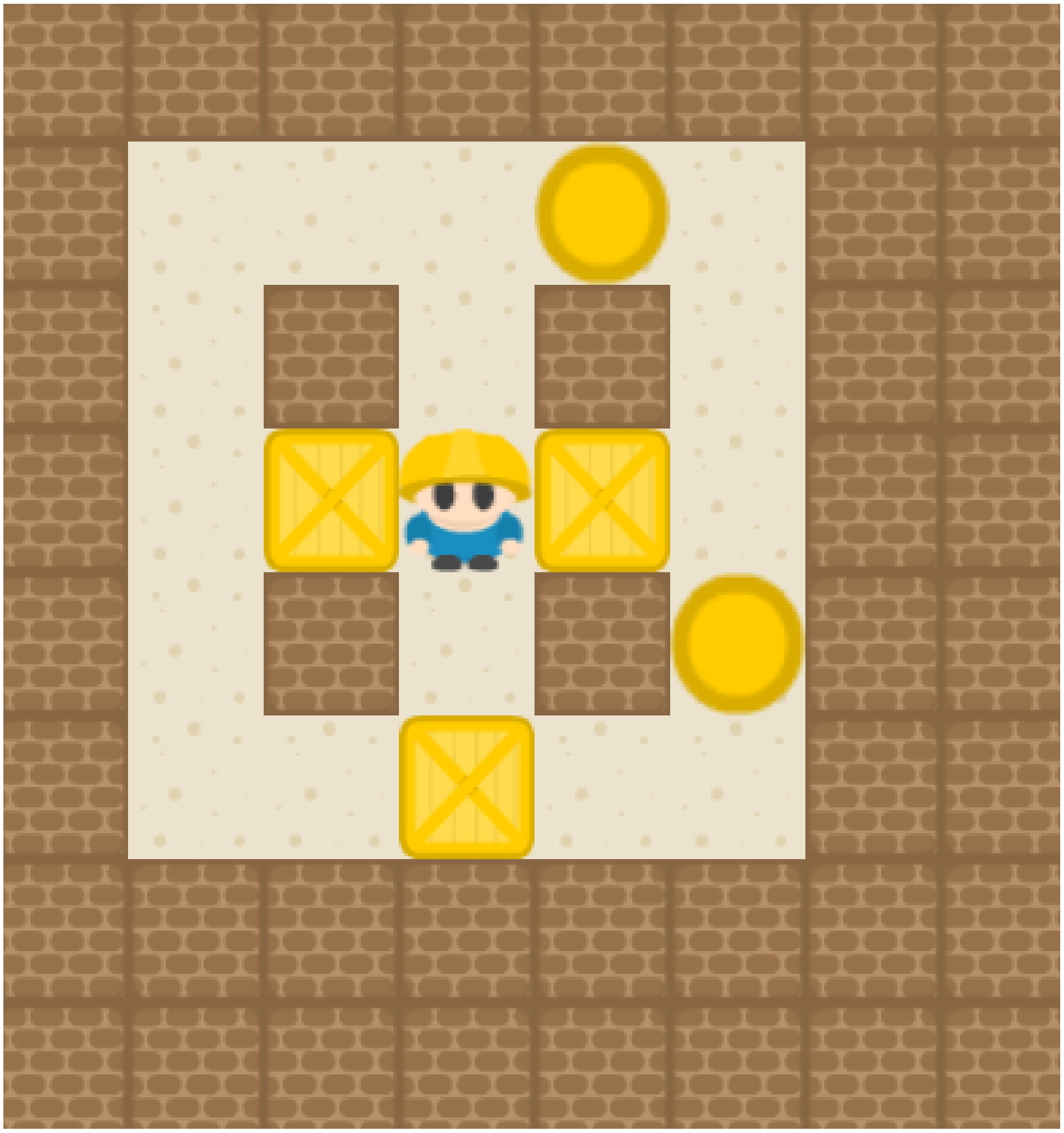
1- Menu:



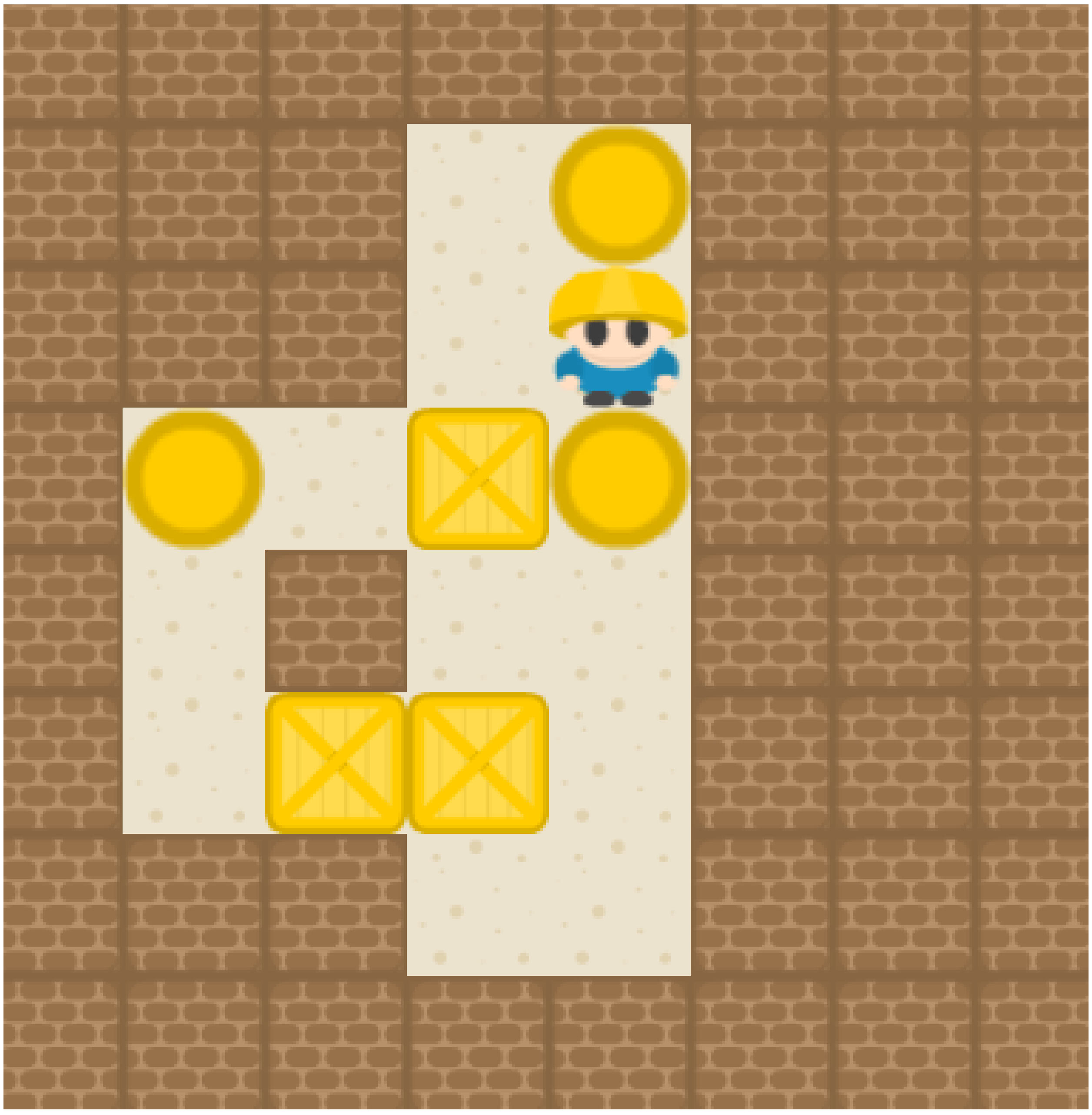
2- Level 1:



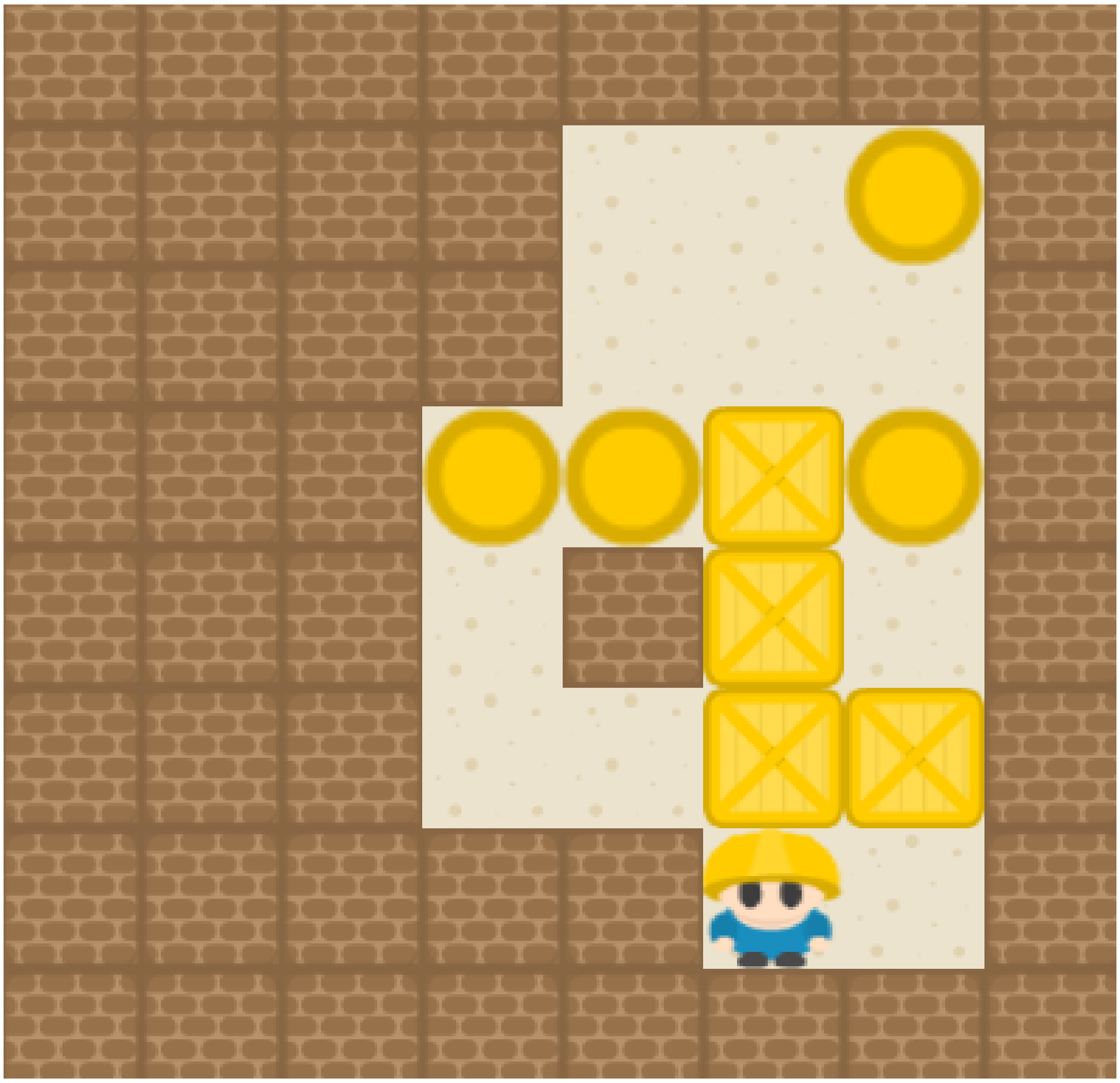
3- Level 2:



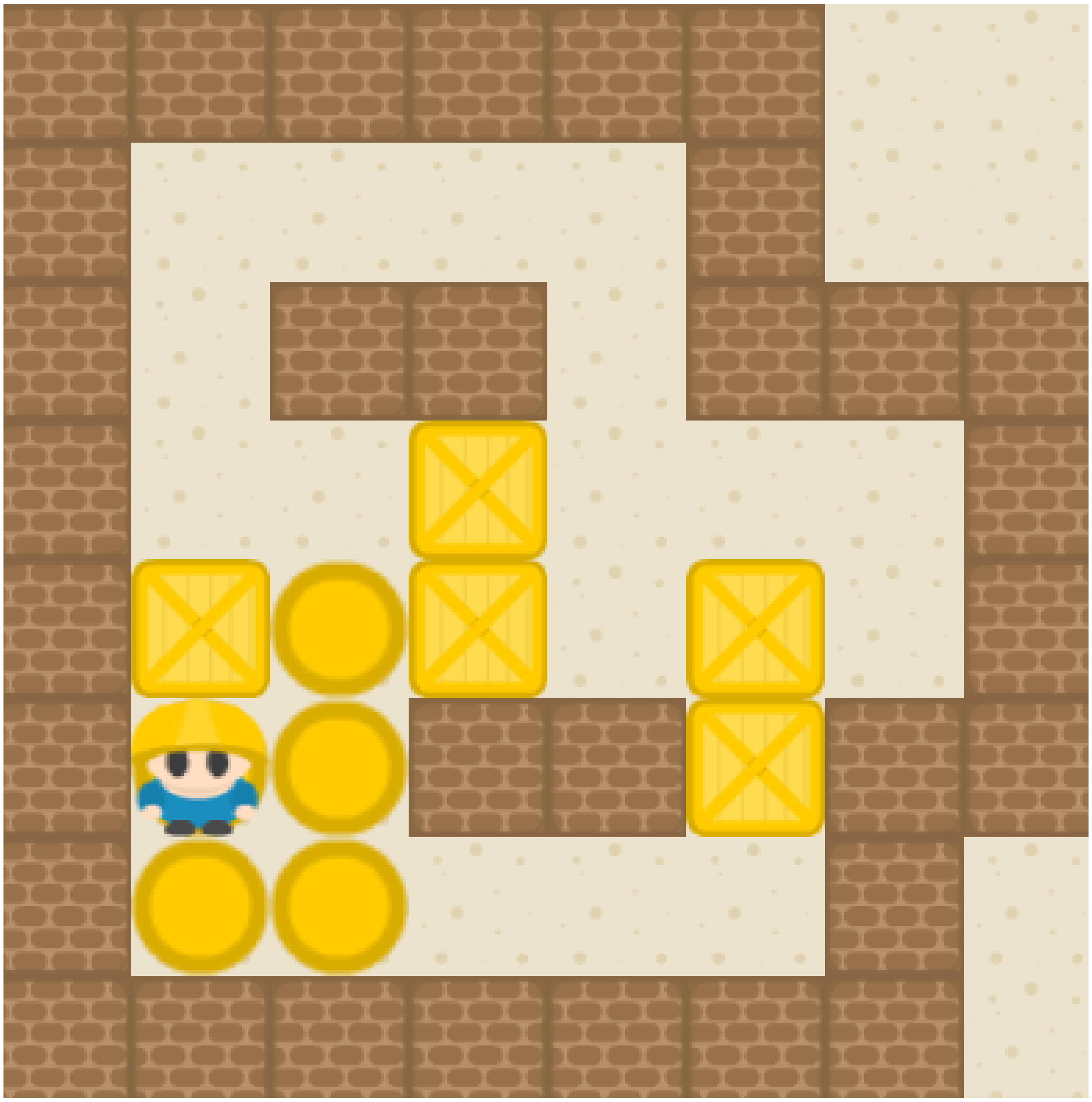
4- Level 3:



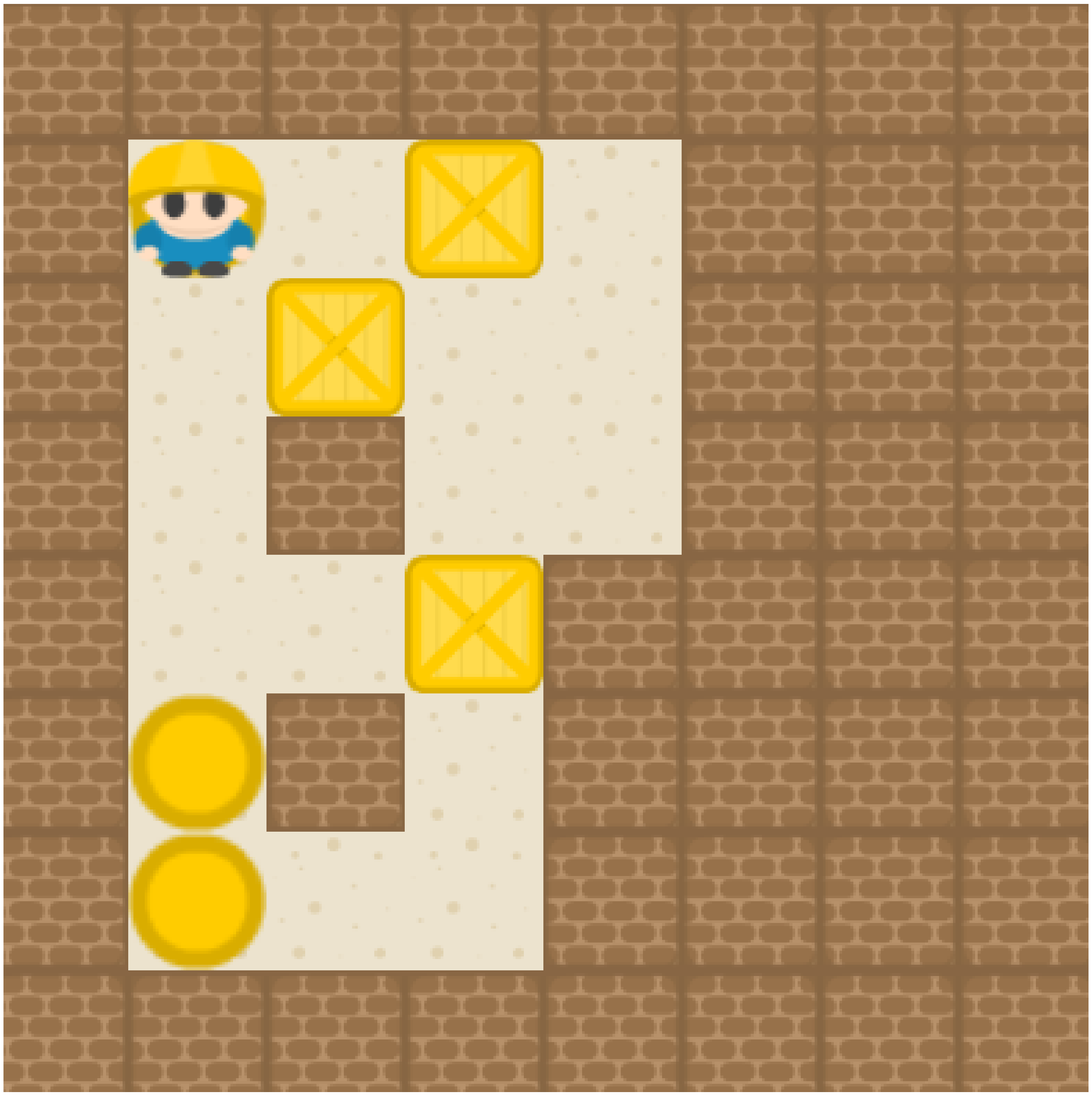
5- Level 4:



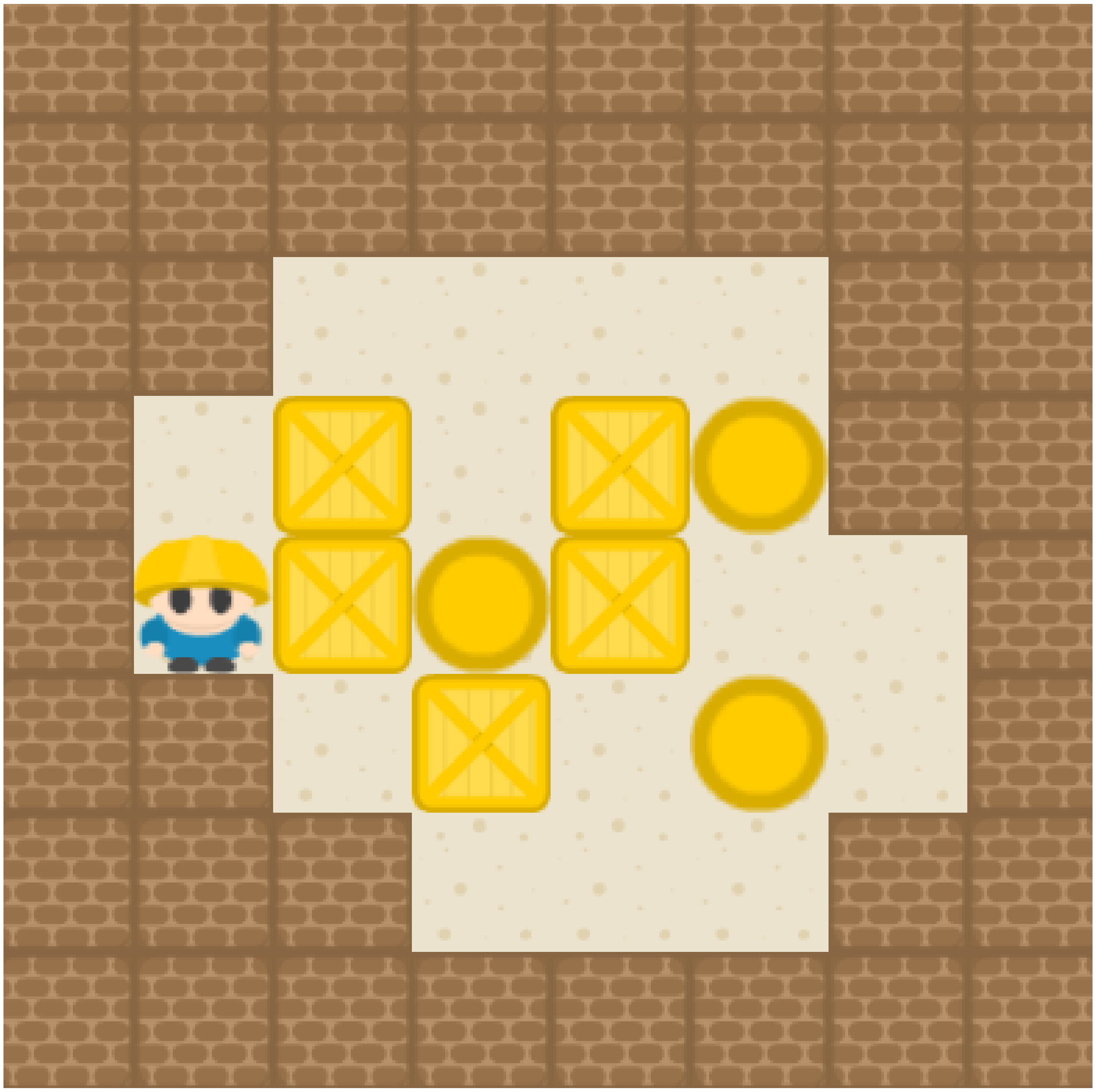
6- Level 5:



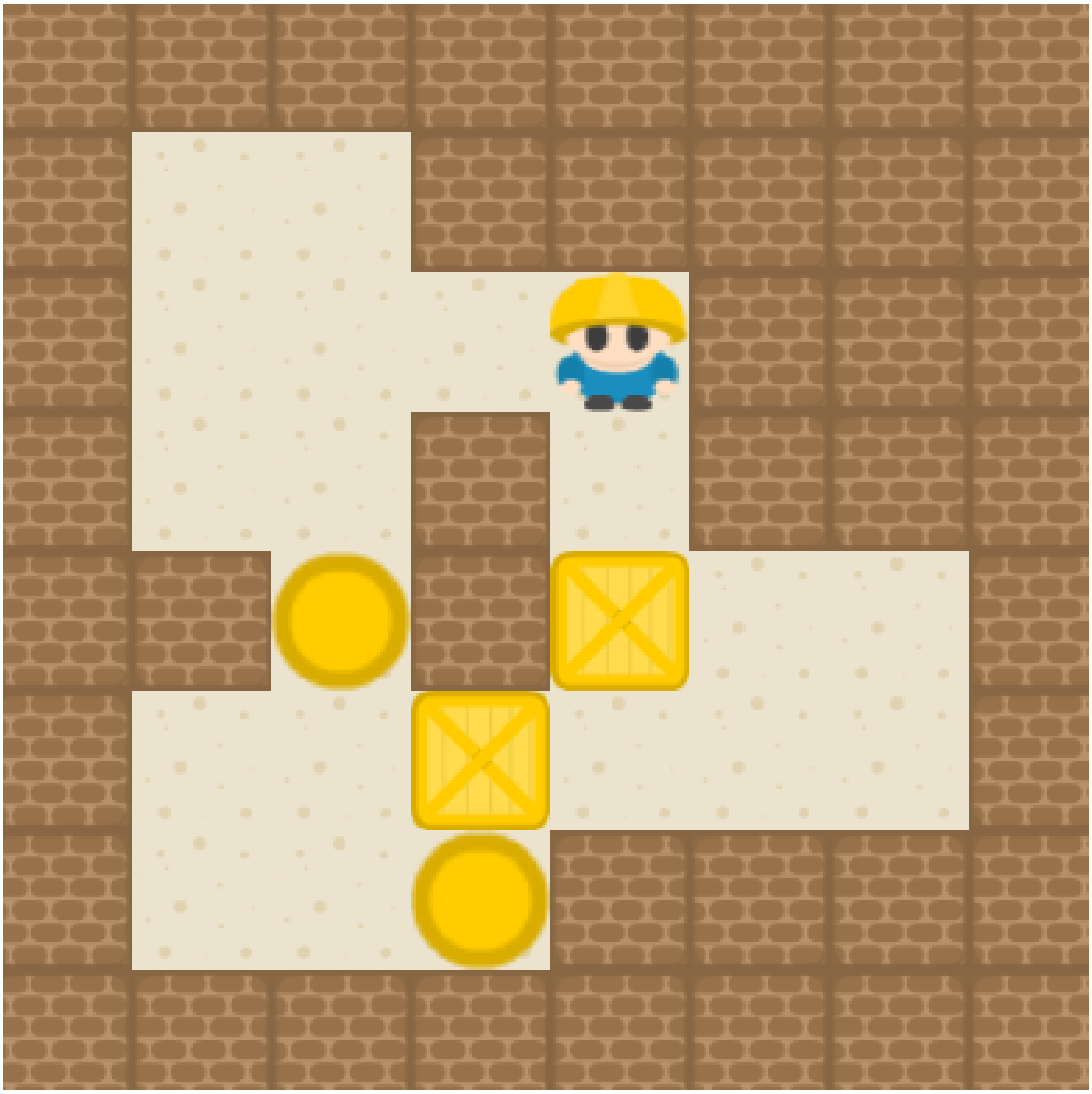
7- Level 6:



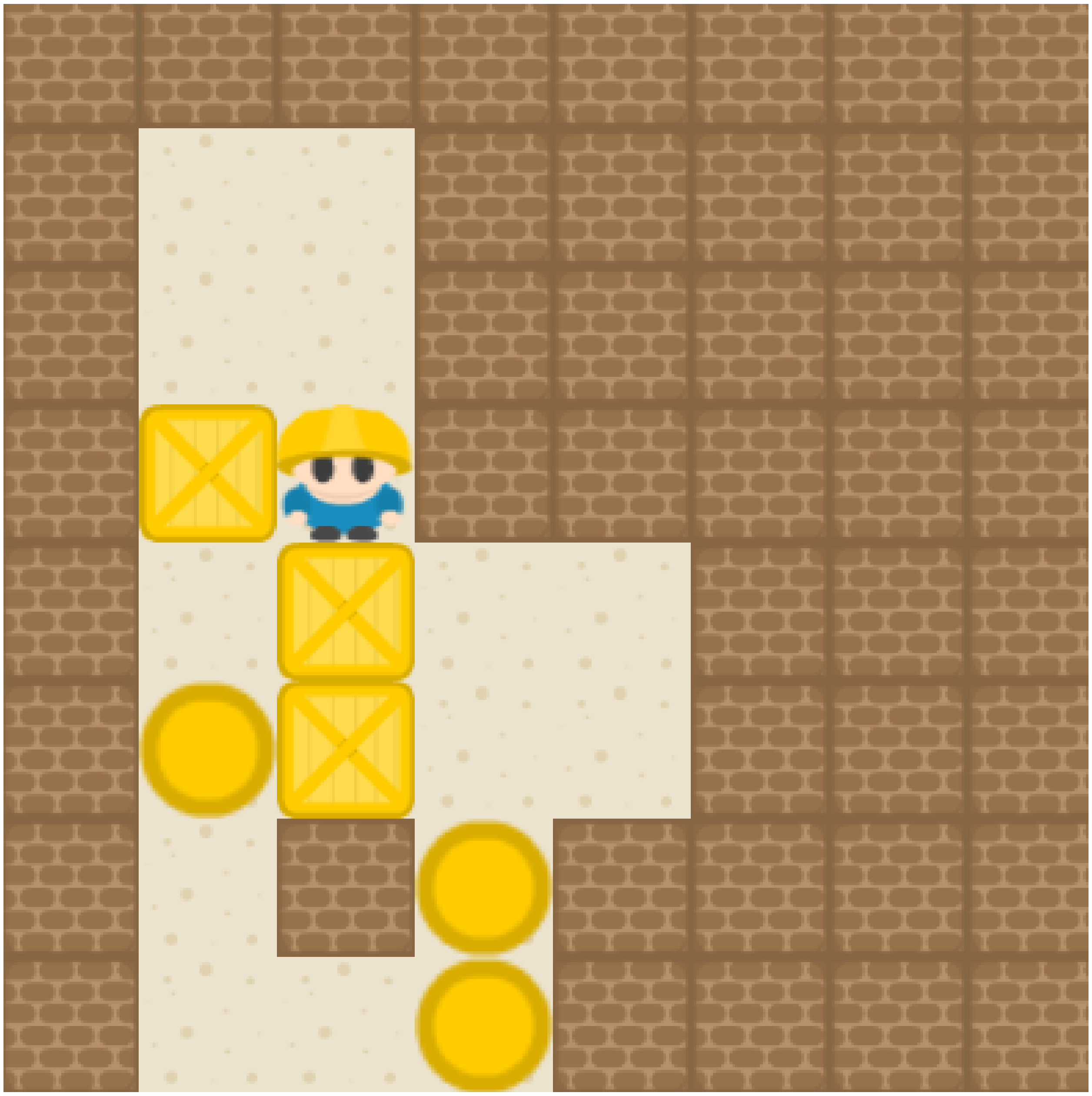
8- Level 7:



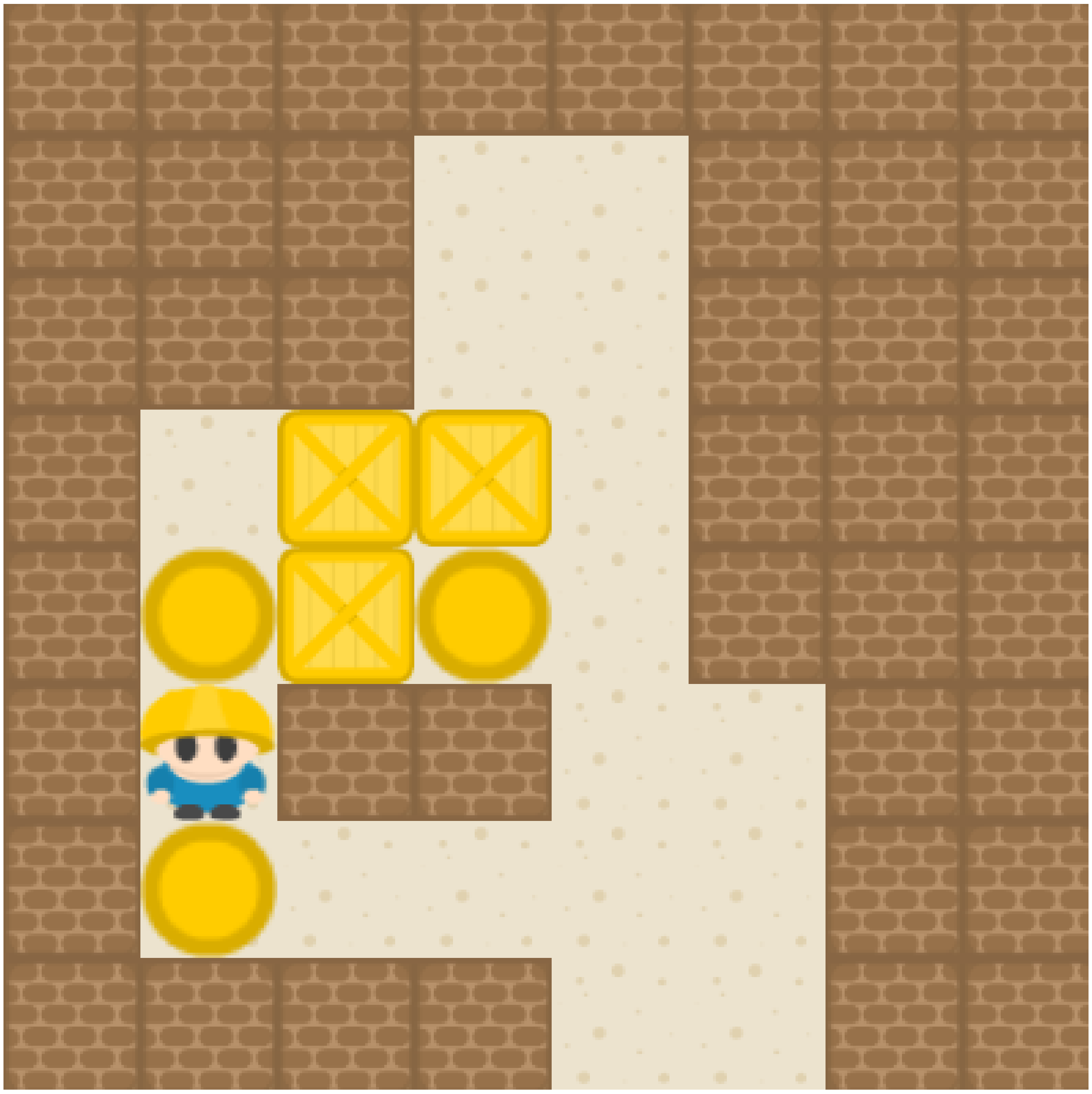
9- Level 8:



10- Level 9:



11- Level 10:



Future considerations:

- Add hints button to the right menu of each level.
- Increase number of levels.
- Add different visualizations and leave the choice to the user to choose the visualization he / she feels comfortable with.
- Add solutions to levels.
- Add timer.
- Add music when playing.
- Add redo button.
- Add statistics button.
- Add help button.

Conclusion:

This project was a good experience for learning new skills. We're looking further to develop more games and, perhaps, more programs that might benefit society from different sides. Finally, coding is passion and not a job so we should love our work so that we can give our maximum and develop the best. Be creative, be yourself.

Total number of words: 4550.