Examen 1 matin

Lorsque le jeu se lance, le système numérote aléatoirement les cibles. Le but du jeu est alors de placer chaque caisse sur la cible portant le même numéro.

Un bouton permet de renuméroter (aléatoirement) les cibles. Chaque renumération "coûte" 10 mouvements.

```
public class Goal extends Element{
    private int id;
    public Goal() {
        super(CellValue.GOAL);
    }
    public int getId() {
        return id;
    }

    public void setId(int id) {

        this.id = id;

    }
}
```

mettre un id dans goal comme dans box. Dans cellView4Plat je dois modifier la methode updateView pour faire apparaitre les id sur les goal

```
case GOAL:
    System.out.println("Adding goal image to cell at (" + line + ", " + col
+ ")");
    addImageView(goalImage);
    Goal goal = (Goal) value;
    if (goal.getId() == 0) {
        int newId = Grid4Design.incrementGoalCount();
        goal.setId(newId);
    }
    Label labelgoal = new Label(String.valueOf(goal.getId()));
    labelgoal.setFont(new Font("Arial", 10));
    labelgoal.setTextFill(Color.BLACK);
    //labelgoal.setStyle("-fx-background-color: white; -fx-padding: 5px;");
    getChildren().add(labelgoal);
    break;
```

mais je n'ai pas de methode incrementGoalCount que je dois créer dans Grid4design qui va compter tout les goal se trouvant dans la classe lorsque l'on ajoute des goal

d'abord j'ajoute ceci dans la classe

```
private static int goalCount = 1;
```

ensuite je creer la methode incrementGoalCount

```
public static int incrementGoalCount(){
    System.out.println("goal"+goalCount);
    return goalCount++;
}
```

lors du resetGrid je met goalCount = 1 pour que le nombre de goal soit réinitialisé

```
@Override
public void resetGrid(int newWidth, int newHeight) {
    this.gridWidth.set(newWidth);
    this.gridHeight.set(newHeight);
    this.cell4Design = new Cell4Design[newWidth][newHeight];
    boxCount = 1;
    goalCount = 1;
    initializeCells();
    triggerGridChange();
}
```

Je dois aussi creer une methode qui decremente dans grid4Design

```
public static void decrementGoalCount(){
    if(goalCount>1){
        goalCount--;
    }
}
```

lorsque je supprime un objet

```
@Override
public void deleteObject() {
    if (!isEmpty()) {
        Cell cell = board.getGrid().getMatrix()[line][col];
        if (cell.getValue().stream().anyMatch(e -> e instanceof Box)) {
            Grid4Design.decrementBoxCount(); // Decrement the box count
        }else if(cell.getValue().stream().anyMatch(e -> e instanceof
Goal)){
            Grid4Design.decrementGoalCount();
        }
        cell.clearValues();
        board.getGrid().triggerGridChange(); // cela a permis la mise a
jour lors de la suppression lors du drag
    }
}
```

Dans ce cas la si j'ajoute 3 goal dans la map d'un coup elles auront toute un id de 1 donc j'ai ajouter ce code dans cette methode addObject dans CellViewModel4Design

```
@Override
public void addObject() {
    Element selectedTool = boardViewModel.getSelectedTool();

    System.out.println("Attempting to add: " + selectedTool.getType());

    // Check for the current presence of a player and handle the unique
rules for a player
    if (selectedTool.getType() == CellValue.PLAYER &&
boardViewModel.hasPlayer()) {
        System.out.println("A player is already present on the grid. Cannot
```

```
add another.");
            return;
        }

        // Check if adding a box, replace with a new instance (though this may
be unnecessary unless the Box class contains unique states)
        if (selectedTool.getType() == CellValue.BOX) {
            Box newBox = new Box();
            selectedTool = newBox; // Update selectedTool with a new Box
instance
        }
        if (selectedTool.getType() == CellValue.GOAL) {
            Goal newGoal = new Goal();
            selectedTool = newGoal; // Update selectedTool with a new Box
instance
        }

        System.out.println("Is cell empty? " + isEmpty());

        // Ensure we update the cell value correctly, allowing goal to combine
with box/player
        if (selectedTool.getType() != CellValue.EMPTY) {
            updateCellValue(selectedTool);
        }
    }
```

Maintenant qu'au niveau de la vue c'est fait je vais faire en sorte de faire en sorte que lorsqu'une box qui est sur le goal et que les 2 ont le même id et bien le score s'incremente.

Tout ce passe dans la methode MovePlayer dans Board4Play

Voici la methode movePlayer qui va prendre en compte boxId==goalId

```
public static void movePlayer(Direction direction) {

    if (boxesOnGoals == grid4Play.getTargetCount()) {
        System.out.println("All boxes are on the goals. No more moves
allowed.");
        return;
    }

    int[] playerPosition = grid4Play.findPlayerPosition();
    if (playerPosition == null) {
        System.out.println("Player not found.");
        return;
    }

    Cell oldCell =
grid4Play.getMatrix()[playerPosition[0]][playerPosition[1]];
    boolean isPlayerOnGoal = oldCell.hasElementOfType(Goal.class);
    Goal existingGoal = null;
    if (isPlayerOnGoal) {
        existingGoal = (Goal) oldCell.getElementOfType(Goal.class);
    }
    int newRow = playerPosition[0] + direction.getDeltaRow();
    int newCol = playerPosition[1] + direction.getDeltaCol();
```

```
    if (!isMoveValid(newRow, newCol, direction)) {
        System.out.println("Move to " + newRow + ", " + newCol + " is
invalid.");
        return;
    }

    Cell targetCell = grid4Play.getMatrix()[newRow][newCol];
    if (targetCell.hasElementOfType(Box.class)) {
        int boxNewRow = newRow + direction.getDeltaRow();
        int boxNewCol = newCol + direction.getDeltaCol();

        Box box = (Box) targetCell.getElementOfType(Box.class);
        boolean isBoxOnGoal = targetCell.hasElementOfType(Goal.class);
        Goal goalInOldCell = isBoxOnGoal ? (Goal)
targetCell.getElementOfType(Goal.class) : null;
        boolean willBoxBeOnGoal =
grid4Play.getMatrix()[boxNewRow][boxNewCol].hasElementOfType(Goal.class);
        Goal goalInNewCell = willBoxBeOnGoal ? (Goal)
grid4Play.getMatrix()[boxNewRow][boxNewCol].getElementOfType(Goal.class) :
null;

        if (isPositionValid(boxNewRow, boxNewCol)) {
            // Retrieve the existing Box object from the targetCell

            // Déplace la boîte vers la nouvelle cellule
            grid4Play.play(boxNewRow, boxNewCol, box);
            if (goalInNewCell != null && box.getId() ==
goalInNewCell.getId()) {
                boxesOnGoals++; // Incrémentez si les IDs correspondent
            }
            if (isBoxOnGoal && !willBoxBeOnGoal && goalInOldCell != null &&
box.getId() == goalInOldCell.getId()) {
                boxesOnGoals--; // Décrémentez si la boîte quitte un goal
avec le même ID
            }
            if (!isBoxOnGoal) {
                grid4Play.play(newRow, newCol,
createElementFromCellValue(CellValue.EMPTY));
            }
            if (boxesOnGoals == grid4Play.getTargetCount()) {


BoardViewModel4Play.getBoardView4Play().displayYouWinLabel(moveCount);
            }
            grid4Play.addPlayerToCell(newRow, newCol);

BoardViewModel4Play.getBoardView4Play().updateGoalsReached(boxesOnGoals);
            lastMoveWasSuccessful = true;
        } else {
            System.out.println("Invalid move: Box cannot be moved to (" +
boxNewRow + ", " + boxNewCol + ")");
            return;
        }
    } else {
        grid4Play.play(newRow, newCol,
createElementFromCellValue(CellValue.PLAYER));
    }
    // If the player is moving from a goal, keep the goal.
    if (isPlayerOnGoal) {
        grid4Play.play(playerPosition[0], playerPosition[1], existingGoal);
```

```
    } else {
        grid4Play.play(playerPosition[0], playerPosition[1],
createElementFromCellValue(CellValue.GROUND));

    }
    System.out.println(lastMoveWasSuccessful);

    moveCount++;
    BoardViewModel4Play.getBoardView4Play().updateMovesLabel(moveCount);



}
```

au niveau des modif que j'ai fait la 1ere est qu'a chaque fois que je passe sur un goal et bien ca me créer un nouveau Goal, pour le coup j'ai modifier ceci pour garder un goalExistant.

Ensuite introduit une vérification d'ID lorsque les boîtes se déplacent sur ou hors des goals, ce qui permet un suivi précis et correct des scores basé sur des correspondances d'ID spécifiques entre les boîtes et les goals.

Mais pour le moment le undo et redo ne fonctionne pas correctement.

Maintenant j'ai besoin d'ajouter un un bouton qui va apparaitre dans le play et une fois ou je clique dessus les id des goal se melange

Pour la création du nouveau bouton j'ai modifier la methode qui prenait en compte le bouton finish et j'ai ajouter ceci

```
private void createControlButtons() {
    // Création du bouton Finish
    Button finishButton = new Button("Finish");
    finishButton.setOnAction(event -> {
        boardViewModel4Play.getBoard4Play().setMoveCount(0);
        BoardView4Play.updateMovesLabel(0);
        Stage currentStage = (Stage) this.getScene().getWindow();
        currentStage.close();
        // Supposons que vous souhaitez réouvrir une certaine vue ou faire
quelque chose de spécifique après la fermeture
        Stage stage = new Stage();
        new BoardView4Design(stage, new
BoardViewModel4Design(boardViewModel4Play.getBoard()));
    });

    // Création du bouton Numbering Target
    Button numberingTargetButton = new Button("Numbering Target");
    numberingTargetButton.setOnAction(event -> {
        // Action à réaliser quand le bouton Numbering Target est pressé
        System.out.println("Numbering Target button pressed!");
        // Vous pouvez ajouter ici la logique spécifique pour le traitement
des objectifs numérotés
    });

    // Ajout des boutons à un conteneur HBox pour un alignement horizontal
    HBox buttonContainer = new HBox(10); // Espacement de 10 pixels entre
les boutons
    buttonContainer.setAlignment(Pos.CENTER);
    buttonContainer.getChildren().addAll(finishButton,
numberingTargetButton);
```

```
    // Ajout du conteneur de boutons à la vue
    setBottom(buttonContainer);
}
```

et je l'ai mit dans le constructeur

```
public BoardView4Play(Stage primaryStage, BoardViewModel4Play
boardViewModel4Play) {


    this.boardViewModel4Play = boardViewModel4Play;
    initializeTotalGoals(boardViewModel4Play.getTargetCount());
    updateGoalsReached(boardViewModel4Play.getGoalsReached());
    start(primaryStage);
    createControlButtons();
    //createReplay();
    youWinLabel.setVisible(false);

    createHeader();
    configMainComponents(primaryStage);
    createGrid();


}
```

Examen 2 après midi

Pour les murs qui deviennent traversable j'ai créer un nouvel element WallTraversable et un nouvel enum TRAVERSABLE que j'ai ajouter a la classe WallTraversable

```
package sokoban.model.element;

import sokoban.model.CellValue;

public class WallTraversable extends Element{
    public WallTraversable() {
        super(CellValue.TRAVERSABLE);
    }
}
```

```
public enum CellValue {
    EMPTY, PLAYER, BOX, GOAL, GROUND, WALL,TRAVERSABLE
}
```

Dans la methode Grid4Play, j'ai fait en sorte que lorsqu'on initialise la grille avec ses cellule via la methode initializeCells, j'ai modifié cette dernière pour qu'elle devienne un int au lieu d'un void j'initialise un compteur et j'ai pris cellule par cellule s'il y a un wall dedans et j'incremente le compteur

```
private int initializeCells(Grid4Design grid4Design) {
    int wallCount = 0;
    for (int i = 0; i < gridWidth.get(); i++) {
        for (int j = 0; j < gridHeight.get(); j++) {
            this.cell4Play[i][j] = new
Cell4play(grid4Design.getCell4Design(i,j)); // Ou une autre logique pour
initialiser chaque cellule
            Cell cell = grid4Design.getMatrix()[i][j];
            if (cell.getValue().contains( new Wall())) {
                wallCount++;
            }
        }
    }
    return wallCount;
}
```

Apres avoir fait cela , je cree une methode qui va permettre de rendre random le nombre de wall en wallTraversable

```
public void transformWallsToTraversable( int wallCount) {
    Random rand = new Random();
    int traversableCount =wallCount /10;
    while (traversableCount != 0) {
        int randRow = rand.nextInt(gridWidth.get());
        int randCol = rand.nextInt(gridHeight.get());

        Cell cell = getMatrix()[randRow][randCol];
        if (cell.getValue().contains(new Wall())) { // verifie si la
cellule contient un wall
            cell.play(new WallTraversable());
            traversableCount--;
        }
    }
}
```

puis j’appelle les 2 methodes dans le constructeur

```
public Grid4Play(Grid4Design grid4Design) {
    this.gridWidth.set(grid4Design.getGridWidth());
    this.gridHeight.set(grid4Design.getGridHeight());
    this.cell4Play = new Cell4play[gridWidth.get()][gridHeight.get()];
    int wallcount = initializeCells(grid4Design);
    System.out.println("Grid Width: " + this.gridWidth.get());
    System.out.println("Grid Height: " + this.gridHeight.get());
    transformWallsToTraversable(wallcount);
}
```

ensuite dans la classe Cell dans la methode AddValue , j’ajoute le WallTraversable de façon a ce qu’il soit ajouter dans la grille. si l'élément est de type GROUND ou WALL, ou TRAVERSABLE la liste toolObject est d'abord vidée avant d'ajouter l'élément.

```
public void addValue(Element value) {
    // If the incoming element is Ground or Wall, clear the list and add
only this element
    if (value.getType() == CellValue.GROUND || value.getType() ==
CellValue.WALL || value.getType() == CellValue.TRAVERSABLE) {
        toolObject.clear();
```

```
        toolObject.add(value);
    } else if (value.getType() == CellValue.BOX || value.getType() ==
CellValue.PLAYER) {
        // If a Goal already exists, add the new element without removing
the Goal
        if (hasElementOfType(Goal.class)) {
            Element goal = getElementOfType(Goal.class);
            toolObject.clear();
            toolObject.add(value);
            toolObject.add(goal);

        } else if(hasElementOfType(WallTraversable.class)) {
            Element walltraversable =
getElementOfType(WallTraversable.class);
            toolObject.clear();
            toolObject.add(value);
            toolObject.add(walltraversable);

        }else {
            toolObject.clear();
            toolObject.add(value);
        }
    } else if (value.getType() == CellValue.GOAL) {
        // If adding a Goal, check if a Box or Player is present
        Element box = getElementOfType(Box.class);
        Element player = getElementOfType(Player.class);
        toolObject.clear();
        if (box != null) toolObject.add(box);
        if (player != null) toolObject.add(player);
        toolObject.add(value);
    }
}
```

puis dans le Board4Play j'ai du modifié 2 methode. MovePlayer et IsMoveValide

d'abord j'ai modifié IsMoveValide de façon a ce que le joueur puisse passer quand c'est un wallTraversable comme si la cellule était vide ou s'il y avais un goal.

```
private static boolean isMoveValid(int newRow, int newCol, Direction
direction) {
    System.out.println("Checking move validity for: " + newRow + ", " +
newCol);


    if (newRow < 0 || newRow >= grid4Play.getGridWidth() || newCol < 0 ||
newCol >= grid4Play.getGridHeight()) {
        System.out.println("Move is invalid: Player out of bounds.");
        return false;
    }

    Cell targetCell = grid4Play.getMatrix()[newRow][newCol];
    System.out.println("Target cell value: " + targetCell.getValue());


    if (targetCell.isEmpty() || targetCell.hasElementOfType(Goal.class)||
targetCell.hasElementOfType(WallTraversable.class)) {
        return true;
    }

    if (targetCell.hasElementOfType(Box.class)) {
```

```
        int boxNewRow = newRow + direction.getDeltaRow();
        int boxNewCol = newCol + direction.getDeltaCol();

        System.out.println("Checking next cell for box at: " + boxNewRow +
", " + boxNewCol);


        if (boxNewRow < 0 || boxNewRow >= grid4Play.getGridWidth() ||
boxNewCol < 0 || boxNewCol >= grid4Play.getGridHeight()) {
            System.out.println("Move is invalid: Box out of bounds.");
            return false;
        }

        Cell boxNextCell = grid4Play.getMatrix()[boxNewRow][boxNewCol];
        System.out.println("Next cell value for box: " +
boxNextCell.getValue());


        boolean canMoveBox = (boxNextCell.isEmpty() ||
boxNextCell.hasElementOfType(Goal.class)) &&
!boxNextCell.hasElementOfType(Box.class);
        System.out.println("Can move box: " + canMoveBox);
        return canMoveBox;
    }

    return false;
}
```

puis dans MovePlayer j'ai ajouter un boolean de façon a ce que quand le joueur traverse le mur traversable et bien le mur reste toujours la et qu il ne soit pas remplacer par un ground

```
public static void movePlayer(Direction direction) {

    if (boxesOnGoals == grid4Play.getTargetCount()) {
        System.out.println("All boxes are on the goals. No more moves
allowed.");
        return;
    }

    int[] playerPosition = grid4Play.findPlayerPosition();
    if (playerPosition == null) {
        System.out.println("Player not found.");
        return;
    }

    Cell oldCell =
grid4Play.getMatrix()[playerPosition[0]][playerPosition[1]];
    boolean isPlayerOnGoal = oldCell.hasElementOfType(Goal.class);
    boolean isPlayerOnTraversabe =
oldCell.hasElementOfType(WallTraversable.class);

    int newRow = playerPosition[0] + direction.getDeltaRow();
    int newCol = playerPosition[1] + direction.getDeltaCol();


    if (!isMoveValid(newRow, newCol, direction)) {
        System.out.println("Move to " + newRow + ", " + newCol + " is
invalid.");
        return;
    }
```

```
    Cell targetCell = grid4Play.getMatrix()[newRow][newCol];
    if (targetCell.hasElementOfType(Box.class)) {
        int boxNewRow = newRow + direction.getDeltaRow();
        int boxNewCol = newCol + direction.getDeltaCol();

        boolean isBoxOnGoal = targetCell.hasElementOfType(Goal.class);
        boolean willBoxBeOnGoal =
grid4Play.getMatrix()[boxNewRow][boxNewCol].hasElementOfType(Goal.class);

        if (isPositionValid(boxNewRow, boxNewCol)) {
            // Retrieve the existing Box object from the targetCell
            Box box = (Box) targetCell.getElementOfType(Box.class);
            // Move the existing Box object to the new cell
            grid4Play.play(boxNewRow, boxNewCol, box);
            if (!isBoxOnGoal && willBoxBeOnGoal) {
                boxesOnGoals++; // Increment the number of boxes on goals
            }
            if (isBoxOnGoal && !willBoxBeOnGoal) {
                boxesOnGoals--; // Decrement the number of boxes on goals
            }
            if (!isBoxOnGoal) {
                grid4Play.play(newRow, newCol,
createElementFromCellValue(CellValue.EMPTY));
            }
            if (boxesOnGoals == grid4Play.getTargetCount()) {


BoardViewModel4Play.getBoardView4Play().displayYouWinLabel(moveCount);
            }
            grid4Play.addPlayerToCell(newRow, newCol);

BoardViewModel4Play.getBoardView4Play().updateGoalsReached(boxesOnGoals);
            lastMoveWasSuccessful = true;
        } else {
            System.out.println("Invalid move: Box cannot be moved to (" +
boxNewRow + ", " + boxNewCol + ")");
            return;
        }
    } else {
        grid4Play.play(newRow, newCol,
createElementFromCellValue(CellValue.PLAYER));
    }
    // If the player is moving from a goal, keep the goal.
    if (isPlayerOnGoal) {
        grid4Play.play(playerPosition[0], playerPosition[1], new Goal());
    } else if (isPlayerOnTraversabe) {
        grid4Play.play(playerPosition[0], playerPosition[1], new
WallTraversable());
    } else {
        grid4Play.play(playerPosition[0], playerPosition[1], new Ground());
    }
    lastMoveWasSuccessful = true;
    System.out.println(lastMoveWasSuccessful);

    moveCount++;
    BoardViewModel4Play.getBoardView4Play().updateMovesLabel(moveCount);



}
```

Concernant le bouton qui permet de cacher ou de faire apparaitre les murs traversable.

D'abord dans BoardViewModel4Play j'ai ajouté la propriété showTraversableWalls

```
private static final BooleanProperty showTraversableWalls = new
SimpleBooleanProperty(true);
```

Cette propriété boolean indique si les murs traversables doivent afficher un "X" ou non. Elle est initialisée à true pour montrer les "X" par défaut.

Par la suite j'ai créée des méthodes associer a ce booleanProperty

```
public static BooleanProperty showTraversableWallsProperty() {
    return showTraversableWalls;
}
public void ToggleshowTraversableWallsProperty(){
    CellViewModel4Play.setShowTraversable();
    showTraversableWalls.set(!showTraversableWalls.get());
}
```

showTraversableWallsProperty() : Cette méthode retourne la propriété showTraversableWalls. Cela permet à d'autres parties du code de se lier à cette propriété et de réagir à ses changements.

ToggleshowTraversableWallsProperty() : Cette méthode inverse la valeur de showTraversableWalls. Elle appelle également une méthode statique de CellViewModel4Play pour synchroniser l'état de cette propriété.

Ensuite dans **CellViewModel4Play**

J'ajoute aussi une propriété

```
private static BooleanProperty ShowTraversable = new
SimpleBooleanProperty(true);
```

ainsi que ses méthodes associé

```
public static void setShowTraversable() {
    ShowTraversable.set(!ShowTraversable.get());
}
public BooleanProperty getShowTraversable(){
    return ShowTraversable;
}
```

setShowTraversable() : Cette méthode statique inverse la valeur de ShowTraversable. Cela permet de synchroniser l'état de cette propriété à partir de BoardViewModel4Play

getShowTraversable() : Cette méthode retourne la propriété ShowTraversable. Cela permet à d'autres parties du code de se lier à cette propriété et de réagir à ses changements.

Ensuite dans boardView4Play j'ajoute une methode permettant de créer le bouton pour afficher/masquer les x des murs traversable et je l'ajoute directement dans le constructeur de la classe.

```
private void createToggleTraversableWallsButton() {
    Button toggleTraversableWallsButton = new Button();
    toggleTraversableWallsButton.setFocusTraversable(false);
    toggleTraversableWallsButton.textProperty().bind(

Bindings.when(boardViewModel4Play.showTraversableWallsProperty())
                    .then("Hide Crossable Wall(s)")
                    .otherwise("Show Crossable Wall(s)")
    );


    toggleTraversableWallsButton.setOnAction(event ->
boardViewModel4Play.ToggleshowTraversableWallsProperty());

    playFinishContainer.getChildren().add(toggleTraversableWallsButton);
}
```

createToggleTraversableWallsButton() : Cette méthode crée un bouton permettant d'afficher ou de masquer les "X" sur les murs traversables. Le texte du bouton change dynamiquement en fonction de l'état de showTraversableWalls grâce à une liaison bidirectionnelle. Lorsque le bouton est cliqué, il appelle ToggleshowTraversableWallsProperty() pour inverser l'état de showTraversableWalls.

Dans CellView4Play j'ajoute un Listener pour réagir au changement de ShowTraversable

```
CellView4Play(CellViewModel4Play cellViewModel4Play, DoubleBinding
sizeProperty, GridPane gridPane, int line, int col) {
    this. cellViewModel4Play = cellViewModel4Play;
    this.sizeProperty = sizeProperty;
    this.line = line;
    this.col = col;
    this.gridPane = gridPane;
    setAlignment(Pos.CENTER);

    layoutControls();
    configureBindings();
    cellViewModel4Play.getShowTraversable().addListener((obs, oldVal,
newVal) -> updateView(cellViewModel4Play.valueProperty()));


    cellViewModel4Play.valueProperty().addListener((obs, oldVal, newVal) ->
updateView(newVal));
}
```

Listener sur getShowTraversable() : Ce listener surveille les changements de la propriété ShowTraversable. Lorsqu'elle change, il appelle updateView() pour mettre à jour la vue de la cellule en conséquence.

Puis une mise a jour de la vue dans UpdateView j'ai mit les condition du changement de visuel des murs traversable

```
case TRAVERSABLE:
    addImageView(wallImage);
```

```
    if (cellViewModel4Play.getShowTraversable().get()) {
        Label xLabel = new Label("X");
        xLabel.setFont(new Font("Arial", 20));
        xLabel.setTextFill(Color.BLACK);
        xLabel.setStyle("-fx-background-color: white; -fx-padding: 2px;");
        getChildren().add(xLabel);
        System.out.println("Reload as been called");
    }
   // System.out.println("Reload as been called");

    break;
```

1. La propriété showTraversableWalls est initialisée à true dans BoardViewModel4Play.

   Un bouton pour afficher/masquer les "X" est créé dans BoardView4Play

2. L'utilisateur clique sur le bouton pour afficher/masquer les "X".

   L'action du bouton appelle ToggleshowTraversableWallsProperty() dans BoardViewModel4Play.

3. ToggleshowTraversableWallsProperty() inverse la valeur de showTraversableWalls et appelle CellViewModel4Play.setShowTraversable()

4. CellViewModel4Play.setShowTraversable() inverse la valeur de ShowTraversable.

   Un listener sur ShowTraversable dans CellView4Play détecte ce changement et appelle updateView().

5. updateView() dans CellView4Play met à jour les cellules en fonction de la nouvelle valeur de ShowTraversable.

Toute ces modif pour le moment s'applique aussi pour le mushroom (le début de l'ajout de la fonctionnalité apparition+bouton)

Pour le Mushroom

Il y a la condition qui fait que le joueur ne peux pas bouger tant que le mushroom est visible et bien pour ca j'ai modifié la methode ExecuteMove dans BoardViewModel4Play

```
public void executeMove(Direction direction) {
    if (!showMushroom.get()) {
        System.out.println("Movement blocked because the mushroom is
visible.");
        return; // Ne pas permettre le déplacement si le champignon est
visible
    }
    if (getBoard4Play().canMove(convertDirection(direction))) {
        Command command = new Move(board4Play,
convertDirection(direction));
        commandManager.executeCommand(command);
    }else {
        System.out.println("mouvement impossible");
    }

}
```

j’ai fait en sorte que lorsqu’on clique sur le mushroom le compteur augmente de 20

j’ai d’abord ajouter un event dans le CellView4Play ou je peux cliquer sur les cellule

```
CellView4Play(CellViewModel4Play cellViewModel4Play, DoubleBinding
sizeProperty, GridPane gridPane, int line, int col) {
    this. cellViewModel4Play = cellViewModel4Play;
    this.sizeProperty = sizeProperty;
    this.line = line;
    this.col = col;
    this.gridPane = gridPane;
    setAlignment(Pos.CENTER);

    layoutControls();
    configureBindings();

    cellViewModel4Play.getShowMushroom().addListener((obs, oldVal, newVal)
-> updateView(cellViewModel4Play.valueProperty()));
    cellViewModel4Play.valueProperty().addListener((obs, oldVal, newVal) ->
updateView(newVal));
    setOnMouseClicked(event -> {
        if (event.getClickCount() == 1) {
            handleCellClick();
        }
    });
}
```

Et une methode handleCellClick qui va permettre de gérer l’ajout des mouvement lorsque je clique sur le mushroom . D’ailleur la methode handleCellClick ne fait pas qu’ajouter un nombre de mouvement que l’ont veux, elle fait aussi en sorte que lorsque le mushroom est visible on ne peux pas cliquer sur le mushroom

```
private void handleCellClick() {
    if (cellViewModel4Play.getShowMushroom().get()) {
        ObservableList<Element> elements =
cellViewModel4Play.valueProperty().get();
        boolean containsMushroom = elements.stream().anyMatch(element ->
element.getType() == sokoban.model.CellValue.MUSHROOM);
        if (containsMushroom) {
            cellViewModel4Play.getBoard4Play().addMoves(20);
        }
    }
}
```

dans le Board4Play j’ai creer une methode addMove qui va permettre directement depuis handleCellClick ou depuis n’importe quel autre methode que je vais instancié dans le futur d’augmenter le nombre de mouvement

```
public void addMoves(int additionalMoves) {
    moveCount += additionalMoves;
    BoardViewModel4Play.getBoardView4Play().updateMovesLabel(moveCount);
}
```

Tunnel

J'ai seulement implementer la partie ou le premier tunnel connait le second.

Dans Grid4Play j'ai modifié la methode qui genere les tunnel de manière random dans les cellule vide ainsi que l'ajoute de deux instance

```
private int[] teleporteur1Position = new int[2];
private int[] teleporteur2Position = new int[2];
```

```
public void transformEmptyToTeleporteur(int CellEmptyCount){
    Random random = new Random();
    int TeleporteurCount = 2;
    while(TeleporteurCount !=0){
        int randRow = random.nextInt(gridWidth.get());
        int randCol = random.nextInt(gridHeight.get());
        Cell cell = getMatrix()[randRow][randCol];
        if (cell.isEmpty()) {
            cell.play(new Teleporteur());
            if (TeleporteurCount == 2) {
                teleporteur1Position[0] = randRow;
                teleporteur1Position[1] = randCol;
            } else {
                teleporteur2Position[0] = randRow;
                teleporteur2Position[1] = randCol;
            }
            TeleporteurCount--;
        }
    }
}

public int[] getTeleporteur1Position() {
    return teleporteur1Position;
}

public int[] getTeleporteur2Position() {
    return teleporteur2Position;
}
```

Et j'ai également ajouter 2 getter

Ensuite dans la methode MovePlayer de Board4Play j'ai ajouté ceci à la fin de la methode

```
if (!teleported && targetCell.hasElementOfType(Teleporteur.class)) {
    teleported = true;
    teleportPlayer(new int[]{newRow, newCol});
    teleported = false;
}
```

et pour finir j'ai ajouté la methode teleportPlayer qui va permettre de ramener directement le joueur sur le 2 eme teleporteur depuis la methode movePlayer

```
private static void teleportPlayer(int[] currentPosition) {
    int[] teleporteur1Position = grid4Play.getTeleporteur1Position();
    int[] teleporteur2Position = grid4Play.getTeleporteur2Position();
```

```
    int[] targetPosition = Arrays.equals(currentPosition,
teleporteur1Position) ? teleporteur2Position : teleporteur1Position;



    grid4Play.play(targetPosition[0], targetPosition[1], new Player());

    moveCount++;
    BoardViewModel4Play.getBoardView4Play().updateMovesLabel(moveCount);
}
```

Ajout d'une nouvelle fonctionnalité TRAP and BONUS

Amelioration de l'apparition des element random sans toucher a la methode initializeCell

```
private void transformEmptyToSpecialCells() {
    Random random = new Random();
    int trapCount = 3; // Nombre de pièges à ajouter
    int bonusCount = 2; // Nombre de bonus à ajouter

    // Ajout des pièges aléatoirement
    while (trapCount > 0) {
        int randRow = random.nextInt(gridWidth.get());
        int randCol = random.nextInt(gridHeight.get());
        Cell cell = getCell(randRow, randCol);
        if (cell.isEmpty()) {
            cell.addValue(new Trap());
            trapCount--;
        }
    }

    // Ajout des bonus aléatoirement
    while (bonusCount > 0) {
        int randRow = random.nextInt(gridWidth.get());
        int randCol = random.nextInt(gridHeight.get());
        Cell cell = getCell(randRow, randCol);
        if (cell.isEmpty()) {
            cell.addValue(new Bonus());
            bonusCount--;
        }
    }
}
```

ensuite modification de movePlayer pour ajouter le nombre de mouvement qu'on ajoute lorsque le joueur est sur l'element

```
if (isPlayerOnGoal) {
    grid4Play.play(playerPosition[0], playerPosition[1], new Goal());
    lastMoveWasSuccessful = true;
}else if(isPlayerOnTrap){
    moveCount +=5;
    grid4Play.play(playerPosition[0], playerPosition[1], new Trap());
    //BoardViewModel4Play.getBoardView4Play().updateMovesLabel(moveCount);
}else if (isPlayerOnBonus) {
    moveCount -= 10;
    grid4Play.play(playerPosition[0], playerPosition[1], new Ground());
    //targetCell.clearValues(); // Efface le bonus après utilisation
    lastMoveWasSuccessful = true;
```

```
}else {
    grid4Play.play(playerPosition[0], playerPosition[1], new Ground());
    lastMoveWasSuccessful = true;
}
```

ensuite modification du addValue

```
public void addValue(Element value) {
    // If the incoming element is Ground or Wall, clear the list and add
only this element
    if (value.getType() == CellValue.GROUND || value.getType() ==
CellValue.WALL|| value.getType() == CellValue.TRAP|| value.getType() ==
CellValue.BONUS ) {
        toolObject.clear();
        toolObject.add(value);
    } else if (value.getType() == CellValue.BOX || value.getType() ==
CellValue.PLAYER) {
        // If a Goal already exists, add the new element without removing
the Goal
        if (hasElementOfType(Goal.class)) {
            Element goal = getElementOfType(Goal.class);
            toolObject.clear();
            toolObject.add(value);
            toolObject.add(goal);
        } else if(hasElementOfType(Trap.class)){
            Element trap = getElementOfType(Trap.class);
            toolObject.clear();;
            toolObject.add(trap);
            toolObject.add(value);
        }else if(hasElementOfType(Bonus.class)){
            Element bonus = getElementOfType(Bonus.class);
            toolObject.clear();
            toolObject.add(bonus);
            toolObject.add(value);
        }else {
            toolObject.clear();
            toolObject.add(value);
        }
    } else if (value.getType() == CellValue.GOAL) {
        // If adding a Goal, check if a Box or Player is present
        Element box = getElementOfType(Box.class);
        Element player = getElementOfType(Player.class);
        toolObject.clear();
        if (box != null) toolObject.add(box);
        if (player != null) toolObject.add(player);
        toolObject.add(value);
    }
}
```

et tout le reste comme d’habitude