Analysis for Change Requests

**Task1:** Restrain piece movement for one square straight or diagonally towards the opponent’s home row.

When running the program, piece can be moved to any direction and any position, so related concepts are length of movement, directions, etc.

**Task2:** Implement the capture of a board piece

If a board piece meets another one on a diagonal position, it can replace that one. Besides, we think task 2 is based on completeness of task 1. So in this part, suppose task 1 has been done. Therefore, we need to constrain the movement by two conditions and we may also consider whether to destroy the replaced board piece after the capture.

**Task3:** Implement an undo feature in Menu item “Undo move”

In this request, suppose undo all steps and neither task1 nor task2 has been added in source code. Besides, “undo” should appear as a menu item. This feature seems like a completely new for MiniDraw, so we focus on finding actions of movement first.

**Task4:** Implement a status bar and build it into the framework, update status bar on piece movements

In this task, it definitely requires 2 positions of source code: framework and piece movement. Software maintainers have definite targets to do CL/IA.

---

**Process Record of ArchSummary**

**Task1:**
1. Look at the MICs of MiniDraw Eclipse project. **BoardDrawing** seems related to concepts so select it first. It has five methods. By examining the body of those methods, the movement happens in the body of pieceMovedEvent() but there is no need to change.
2. Check MIRC(BoardDrawing), class **SelectionTool** seems relevant.
3. See MCBI(Figure draggedFigure) and class **BoardFigure** is recommended. BoardFigure has a method performAction() that uses command of type Command to do the movement.
4. In MIRC(BoardFigure), see MCBI(Command command) because this type is an interface and open class **MoveCommand**. In MIM(MoveCommand) there is only one method execute() and invokes an object of type Game to do the actual movement. This method may be impacted.
5. See MCBI(Game game) and open the only recommended class **GameStub**. The method move() is the top ranked MIM and responsible for the piece movement on the board. So this should be impacted and there is no need to check more classes.
Task 2:

1. In request analysis, we already suppose the project has the functionality of task 1. So we mainly focus on operation of selected board piece. Look at the MICs of MiniDraw and choose BoardDrawing to check. In MIM(BoardDrawing), find pieceMovedEvent() (rank 2). It collects two separate lists of objects that are placed on Position from and Position to of a movement. This should be impacted.

We only solve one part of change request. The other part is if a board piece meets another one on a straight position, it cannot move forward. So from MIRC(BoardDrawing), try to find the target classes.

2. Open SelectionTool (rank 2) which seems related. After checking, it is not.

3. BoardFigure appears in MIRC(BoardDrawing) as well as in MIRC(SelectionTool), check it and find the method performAction().

4. Before going deeper, class DragTracker seems more related. Check and find out it is not.

5. Open class BoardActionTool in MIRC(BoardFigure), there is no need to change.

6. Open class FigureChangeEvent, there is no need to change.

7. Return to BoardFigure, see MCBI(Command command) and open class MoveCommand. Method execute() does part of the actual movement.

8. See MCBI(Game game) and open the only recommended class GameStub. The method move() is the top ranked MIM and responsible for the piece movement on the board. So this should be impacted and there is no need to check more classes.

Task 3:

1. In MIC view of project, select BoardDrawing. pieceMovedEvent() is related but there is no need to change.

2. Go to class BoardFigure which is used in pieceMovedEvent(). It has a method performAction() that uses command of type Command to do the movement.

3. In MIRC(BoardFigure), class FigureChangeEvent and BoardActionTool seem relevant. Check them one by one and find there is no need to change them.

4. See MCBI(Command command) because this type is an interface and open class MoveCommand. In MIM(MoveCommand) there is only one method execute() and invokes a method in Game to do the actual movement. There is no need to change MoveCommand.

5. See MCBI(Game game) and open the only recommended class GameStub. The method move() is the top ranked MIM and responsible for the piece movement on the board. So this should be impacted.

6. Then we try to find a class to put the menu item. Go back to MICs of project, this time pick the class that has a name related to view, menu, panel or the other user interface components. Open MiniDrawApplication (rank 2), it is the right place to add menu item of undo feature. Impacted.

Task 4:

1. In MIC view of project, select BoardDrawing. pieceMovedEvent() is related but there is no need to change.

2. Go to class BoardFigure which is used in pieceMovedEvent(). It has a method performAction() that uses command of type Command to do the movement.
3. See MCBI(\textbf{Command} command) because this type is an interface and open class \textbf{MoveCommand}. In MIM(MoveCommand) there is only one method execute() and invokes a method in Game to do the actual movement. There is no need to change \textbf{MoveCommand}.

4. See MCBI(\textbf{Game} game) and open the only recommended class \textbf{GameStub}. The method move() is the top ranked MIM and responsible for the piece movement on the board. So this should be \textbf{impacted}.

5. Return to MIRC(\textbf{BoardFigure}), class \textbf{FigureChangeEvent} and \textbf{BoardActionTool} seem relevant. Check them one by one and find there is no need to change them.

6. Then we try to find the framework. Open \textbf{MiniDrawApplication} (rank 2) in MIC first, showStatus() is related, this class may be \textbf{impacted}.

7. Keep on finding the framework. MiniDrawApplication uses a type Factory, see MCBI(\textbf{Factory} factory) and open \textbf{BreakthroughFactory}. Check it and find it has method createStatusField() which is the target. \textbf{Impacted}.

\textbf{Process Record of JRipples}

\textbf{Task1:}

\textbf{Phase 1:}

1. Main() of MiniDraw is in \textbf{class BreakThroughMain}. After checking, we mark the whole class as \textbf{Propagating}, then class BoardActionTool, DrawingEditor, GameStub, etc. are marked as \textbf{Next}.

2. The name of \textbf{class BoardActionTool} seems related to concepts. Field draggedFigure seems related. Method mouseDrag() relates to the movement of pieces. In mouseUp(), it says “if the figure's associate domain model tell the move is invalid move it back to its starting position”, but it use performAction() from other type to judge the validation, thus mark it as \textbf{Propagating}. Because all methods in BoardActionTool have been checked, we can go to other classes now.

3. Go to \textbf{class BoardFigure} first which has dependencies from mouseUp(). It has isMobile() and performAction() relate to the concept, so mark it as \textbf{Propagating}.

4. Check \textbf{interface Command} next and mark the whole class as \textbf{Propagating}.

5. \textbf{Game} is an interface and its Boolean move() is related to change request, so mark it as \textbf{Propagating}.

6. Check \textbf{class GameStub} and find it implements Game. Mark it as \textbf{Located}.

7. Then check \textbf{class MoveCommand} which has been marked as \textbf{Next} after checking interface Command. In this class, we find execute() which decides the validation of movement. Mark it as \textbf{Located}.

8. Because no new classes are marked as \textbf{Next} after checking interface Game. Pick \textbf{class Position} to check and mark it as \textbf{Unchanged} afterwards.

Because we already find the concepts that where decides the length and directions of movement. There is no need to check more classes in this period.

\textbf{Phase 2:}

1. \textbf{Impact Analysis} begins at classes that are marked as \textbf{Located} in former period. So begin IA from \textbf{class GameStub} and \textbf{MoveCommand}. In this change request, we need to restrain the length and direction of
movement, so predict impacted classes according to this target. Check interface BoardGameObserver first, this time, pieceMovedEvent is not going to modify, so mark it as Unchanged.

2. Check class Constants, interface Game and class Position. They all are not impacted.

**Task2:**

Phase 1:

1. Still begin at class BreakThroughMain, mark it as Propagating. Then check class BoardActionTool, mark it as Propagating.

2. Then go to class BoardFigure, mark it as Propagating because of performAction() and two constructors (relate to UI of board piece).

3. Check interface Command next and mark the whole class as Propagating.

4. Then check class MoveCommand which has been marked as Next after checking interface Command. In this class, mark execute() as Propagating because it ONLY solves the length of movement which has already been changed in task1.

5. Check interface Game next. Because method move() is related to change request, so mark it as Propagating.

6. Check class GameStub and mark move() as Located.

7. Then we check remained classes to find how to destroy a piece, this may belong to a new feature, so we check where those pieces are drawn. Pick class AbstractFigure and invalidate() looks relates to the change request, so mark it as Propagating.

8. Check class BreakthroughPieceFactory, it generates all pieces at the beginning, so mark it as Unchanged.

9. Method remove() in interface Drawing seems related. It is marked as Propagating.

10. Check interface Figure and mark it as Unchanged.

11. In class BoardDrawing (this one is marked as Next after checking class BreakThroughMain), method pieceMovedEvent() operates all the pieces on a certain position of the board. So mark it as Located.

Phase 2:
Check every type marked as Next. No new type is impacted.

**Task3:**

Phase 1:

1. Begin at class BreakThroughMain, mark it as Propagating.

2. Check class BoardActionTool, mark it as Propagating because mouseUp() relates to the end of an action.

3. Go to class BoardFigure first which has dependencies from mouseUp(). Mark it as Propagating after checking the code.

4. Check interface Command next and mark the whole class as Propagating.
5. Then check **class MoveCommand** which has been marked as *Next* after checking interface Command. Mark it as *Propagating*.

6. **Game** is an interface and its Boolean move() is related to change request, so mark it as *Propagating*.

7. Check **class GameStub** and mark it as *Propagating* first because field observer and addObserver() relate to monitoring actions. Then mark move() as *Located* because this is an action.

8. Return to class BreakThroughMain. Go to **class BreakthroughFactory** and mark it as *Propagating*.

9. Return to class BreakThroughMain. Go to **class MiniDrawApplication** and find out it is a good place to put the menu item. Mark it as *Located*.

Phase 2:
Check every type marked as *Next*. No new type is impacted.

**Task4:**

**Phase 1:**
1. Begin at **class BreakThroughMain**, mark whole class as *Propagating*. In order to find the framework, we follow the dependency sequence in source code of this class to check others, that is <Game, GameStub, BreakthroughFactory, DrawingEditor, MiniDrawApplication, BoardDrawing, BoardActionTool, SelectionTool>.

2. Pick **interface Game**, it has method move() but not implements it, so mark it as *Propagating*.

3. Pick **class GameStub**, it realizes method move() and control the Position from and Position to, so mark it as *Located*.

4. Pick **class BreakthroughFactory**, according to createStatusField(), we are sure this is the framework for statusbar, so mark it as *Located*.

5. Pick **interface DrawingEditor**, it has showStatus() and also used in createStatusField() of class BreakthroughFactory, so mark the interface as *Propagating*.

6. Pick **class BoardDrawing**, it has pieceMovedEvent() but not for movement action, so mark it as *Unchanged*.

7. Pick **class MiniDrawApplication**, it has a field statusField and realizes showStatus(), so mark it as *Located*.

Since now, we already locates the positions’ information of move, framework of statusbar and status showing. This period can be finished.

Phase 2:
Check every type marked as *Next*. No new type is impacted.