	Project Grade /20	Feedbacks
s220911	18	The archive does not match the required structured to be compiled with the command given in the statement. Otherwise, the game works perfectly, and detects properly invalid spec. files and prints the precise problem. The code is clear and simple, with good modularity.
s160896-s181386	16	Your program fails to detect some invalid spec. files, which sometimes lead to a crash midgame. Also, the error messages are not always relevant. The game works as expected on valid puzzles. Your code is clear and easy to read. But some methods, especially in the class Board, are not coherent. For instance, the method *Solved()* (which should not start with a capital letter) does not rely on *AtGoal(),* and *movePiece()* can be called without any verification of the validity of the move, when you could have made sure that *ValidMove()* was called before performing the move.
s174501	0	No project -- Archive only contained the .jar file we provided with the statement.
s180389-s2405570	14	The archive does not match the required structured to be compiled with the command given in the statement. The game does not always display square pieces as actual squares in the GUI. Furthermore, the program can crash in the middle of the game. This happens when one tries to move a piece out of the board, and the program prints "No file input given" for no reason. Your code is quite modular but a bit complicated, especially the way the classes Bloc and Grille interact with each other. Try to be coherent when naming your classes, methods and variables, do it all in the same language.
s182398	17	Only give the necessary source files in your submissions, and make sure your code compiles following the exact command given in the statement (the library *sliding-puzzle-gui.jar* was not placed in the right folder for the compilation to work). Your program detects most invalid spec. files, and gives a precise explanation. It also works as expected on valid puzzles. The code is clear and quite modular. I would argue that the PuzzlePiece class should not deal with colors, and let that responsibility entirely to the GUI part of the program. As a side note, an Enum would have been more elegant than Strings to deal with colors. The way you separated the validation of the spec. file and the actual Board construction is also a bit odd, reading the file again could have easily been avoided.
s182498-s211245	15	The archive does not match the required structured to be compiled with the command given in the statement. Your program works well, although it fails to detect some invalid spec. files, e.g., when all the cells are occupied, or when no goals are specified. It also fails to display square pieces as actual squares. The code is quite clear, although some classes could have been merged (PuzzleSpecification and SpecificationFileControl).

s190410	16	The archive does not match the required structured to be compiled with the command given in the statement. Your program works well, although it would have been better to ensure that square pieces are always depicted as actual squares, regardless of the dimension of the grid. The code is clear and quite modular. Some comments would have been welcome.
s191962	16	Your program fails to detect some invalid specification files (when too many pieces or goals are declared for instance) and sometimes prints "null" as a reason for the error. Otherwise your game works well. It is just a bit odd that some pieces start with the "success" color. You were specifically asked in the statement not to use **AWT.** You code is clear. It however lacks some modularity as your Board class involves SlidingPuzzleGUI, and is therefore not independent from the graphical side.
s194923	14	Only give the necessary source files in your submissions, and make sure your code compiles following the exact command given in the statement. Your program fails to detect some invalid spec. files, e.g., when multiple goals are assigned to a single piece. The way your game deals with "diagonal" movement of pieces is a bit odd. I understand what you do, but allowing a piece to pass over its target position is not coherent with how your game works in general. And you should have commented your code to explain this specific behavior. Your code is quite clear. You could have improve the modularity of your code by making your Board (and generally the abstract representation of a puzzle) independent from the graphical interface side.
s195247-s2302062	18	Your program fails to detect some invalid spec. files, e.g., when all the cells are occupied by pieces, or when multiples goals have been defined for the same piece. Furthermore, it usually still throws an exception at the end, instead of cleanly dealing with it. Your game allows the player to go through (or "jump above") another piece. Otherwise, it works quite well, and the goal visualization and the reset button are nice features. The code is quite clear and has good modularity. Good efforts on commenting.

s196087-s221663	9	Your program fails to detect any invalid spec. file correctly. It either starts a clearly invalid game (where some pieces overlap for instance) or just crashes during the game creation. The behavior of the pieces that reach their target position does not comply with the statement requirements (although your solution is clear enough to play). During the game, some pieces can overlap, and even go through other pieces. Your code could have really used some commenting. It a noble effort to try to use inheritance, but your Cell class and its subclasses are quite a complex solution to represent the pieces and the grid. Keeping things simpler would have probably helped you ensure your game behaves correctly.
s201832-s221290	15	Your program does not always detect invalid spec. files, and when it does the error message is not necessarily accurate, and some exceptions are still thrown. Overall the game works well, although it does not close cleanly without an exception when the game is won. The code is quite clear and has good modularity. It could however use more commenting.
s210844	11	Your program should take as input a specification file, not automatically read the one you give in your submission. It should also detect invalid spec. files which it does not at all. The game also terminates as soon a piece reaches its target position, even if other goals are yet to be satisfied. The code is quite clear with good commenting. Some variables are public or package private for no good reasons (in PuzzleReader and Goal for instance). Your code is not very modular either, since your class Board designed to simulate the game already involves both the graphical interface and the spec. file parser.
s212833-s211042	14	Your program wrongfully detects some valid spec. files as invalid (because you swapped *nrows* and *ncols* in FileReader.java). It is also possible to move a piece through another one; when it should actually be blocked. Furthermore it is impossible to close the program properly once the game is won. The code is clear, modular, and some comments help reading it.
s214431-s210880	18	Your game works well. Most invalid spec. files are correctly identified. But it does not close properly, especially after a win when the window is closed, the program keeps running. Square pieces should be displayed as squares. The code is clear and very well commented. Its modularity could be improved by not having the Grid class deal with some graphical aspects (like pixel counts, or RGB colors).

s217327	17	The game works well. The code is quite clear and well commented. There is some modularity, but your classes share too much information between them. You could give the necessary information to the graphical interface, without giving access to all the pieces entirely.
s218152-s226419	18	Your program fails to detect some invalid spec. files, e.g., when all the cells are occupied by pieces. Furthermore, it still throws an exception at the end, instead of simply printing the error and exiting the program. Otherwise the game works as expected, although square pieces are not always displayed as squares. The code is clear, with good modularity.
s220300	11	The archive does not match the required structured to be compiled with the command given in the statement. Your program fails to detect many invalid spec. files (overlapping pieces, no free space, ...). It works as expected on valid puzzles, although square pieces are not always depicted as squares, and it does not close cleanly, but crashes and throws an exception instead. Your code is a bit complex and some commenting would have helped. In particular, why in your graphical interface, you need to store two sets of Pieces, and a PuzzleLogic instance which also contains some Pieces. The structure of your code could have been simpler, and its modularity improved if you tried to segment more the different parts of the program.
s220802-s224392	14	Your program works well and the victory features are a nice addition. It however does not close properly, and throws an exception when the window is closed. Furthermore, on somewhat large grids (15x15), the moves can take a few seconds to be dealt with, which is a bit odd since there should be very little computation to do. The code although well commented, is a bit complex. The data structure you use to represent the pieces is too heavy, hence the inefficiency of your program on larger puzzles. Furthermore, your code is not very modular as your class Game is actually performing almost all the work.
s220915	18	Your program works as expected. The code is clear, although some commenting would have been welcome. It is quite modular, except that you define a color in the Piece class itself, when this should be the responsibility of the graphical part of your code.

s221299-s226111	14	The archive does not match the required structured to be compiled with the command given in the statement. Your program fails to detect some invalid spec. files, e.g., when multiple goals are assigned to a single piece, or when no goals are specified. It works fine on valid files, except that square pieces are not always displayed as squares. The code is clear, but not really modular. The class SlidingPuzzle does almost all the work (apart from parsing the spec. file). It would have been better to deal with the game logic independently from the graphical aspect of your program.
s221605	11	The archive does not match the required structured to be compiled with the command given in the statement. Your program fails to detect many invalid spec. files, and when it does, it still either crashes or open the GUI and gets lock. It works fine on valid puzzles, but the window cannot be closed after victory. Square pieces are not displayed as squares in many puzzles. Your implementation does not really respect the encapsulation principle, as your class PuzzleData gives complete access to all its members (via its *getters* methods) and does not really take its responsibility to simulate the game. It would have been better to develop a more modular program.
s221999-s221223	13	You did not include the expected main class SlidingPuzzle in your archive. I manage to easily build one that calls your classes to initialize and play the game. Always double check your submissions, and try to compile and run your program from the archive you send. Your parser detects almost all invalid spec. files, and give a precise explanation. Your game sometimes crashes when the mouse is released outside the window. The code is clear, well commented. Its modularity could be improved by not making the class Board dependent on DecodeFile for its instantiation. The fact that it was very easy to build a main to test your code saves you a lot of points.
s222337	16	Your program fails to detect some invalid spec. files, e.g., when the pieces cover the whole grid, when no goals are defined, or when multiples goals are assigned to a single piece. Otherwise, it works perfectly on valid puzzles. The code is a bit complex but well commented. It is however not modular. Your class Pieces already involves the graphical interface, when you could have isolated the game simulation from the display.
s225049	0	No project -- Archive only contained the .jar file we provided with the statement.

s225066-s222940	14	Your program fails to detect some invalid spec. files, and when it does sometimes still opens the gui and crashes (when the pieces cover the entire grid). It is possible to move a piece through another one (or jump over it). Square pieces are not always displayed as squares. The code is clear and well commented. It is however not very modular, as your GameManager involves the graphical part. If you tried to have an independent representation of the game, you would have probably noticed the issue with the invalid movement of pieces your code allows.
s225240	16	Your program fails to detect most invalid spec. files. The game works as expected on valid puzzles, although sometimes after victory it prints "Erreur finale : Invalid parameter" before closing. The code is clear and modular. The class PuzzleBoard does not completely respect the encapsulation principle, as it gives access to all the pieces which can be freely moved.
s226324	10	Your program fails to detect many invalid spec. files. You allow any piece to be moved to any empty cell, even if this involve making turns and going through other pieces. Your code is clear and quite modular. But most variables are public for absolutely no reasons, none of your classes respect the encapsulation principle.
s2300594-s2304050	8	Your program fails to detect some invalid spec. files, e.g., when goals are assigned to non-existing pieces, or when multiple goals are assigned to a single piece. It also rejects valid puzzles (like the Klotski one we provide) for no clear reason. On puzzles it validates, it sometimes ends after the first goal is satisfied, not waiting for the others. You also allow diagonal moves that does not respect the game principles at all. The code is clear and there is an effort on commenting. Its modularity could be improved by not making the elementary Pieces deal with the color, but giving that responsibility to the graphical part of your code.
s2300798-s2301323	15	Your program fails to detect any invalid spec. file. Otherwise, it works perfectly on valid puzzles. Your code is clear with some good commenting. Some instance variables are not private for no good reason (especially since you also provide a public getter). Furthermore, your classes do not really respect the encapsulation principle as it is always possible to access most of their internal variables freely.
s2301026-s2300644	14	The archive does not match the required structured to be compiled with the command given in the statement. Your program fails to detect some invalid spec. files, e.g., when the entire grid is covered by pieces. It works as expected on valid puzzles. Your code is clear and very simple. Be careful with the visibility of variables, they should be private in Goal.

s2301027-s2309370	14	Your program fails to detect some invalid spec. files, e.g., when multiples goals are assigned to a single piece, or when no goals are assigned at all. The coloring of the pieces is odd, it seems to change one move after visiting their goal, unless the game is won. The window also does not close properly midgame, and is completely locked after victory. The code could be more modular if your Pieces did not deal with their color directly, but delegated that to the graphical side. The types of movement you used in Piece.move() could have been handled with an Enum.
s2301036-s2300142	20	Your program detects all invalid spec. files and gives a precise explanation. It also works perfectly on valid puzzles. The code is clear and has good modularity.
s2301053-s2301282	16	Your program fails to detect some invalid spec. files, e.g., when multiples goals are assigned to a single piece, or the entire grid is covered by pieces. The game works as expected on valid puzzles, although square pieces are not always displayed as squares. The code is clear and easy to read. It is not very modular as the classes Piece and Board already deal with the graphical interface (managing colors and pixels) instead of simulating the game independently.
s2301108-s220635	14	Your program fails to detect invalid spec. files, e.g., when no goals are declared, or when the entire grid is covered by pieces. Your game seems to have issues when the puzzle admits multiple goals. It either blocks some legitimate movements, or even fails to load valid puzzles. It would have been more elegant to make the SpzlHandler class static, in the sense that it would not need to be instantiated and has a single public class method which simply returns a GameManager instance. The code is clear and well commented. But it is not very modular as the game simulation and the graphical display are completely merged.
s2301154	13	Your game sometimes get completely locked after trying an invalid move. It is also impossible to close the window after victory. The method ReadFile.readFromFile() could have been static, there is no reason to instantiate the class ReadFile. The variables in Piece and Goal should not be public, and why do you also declare setters and getters in that case? The code is quite clear and has good modularity. However, the way you deal with the Grid (as an ArrayList<ArrayList<Cell>>) is perhaps too complicated, and may be the reason why your program sometimes crashes.
s2301647-s2301188	17	You detect most invalid spec. files, and the game works perfectly on valid puzzles. The code is clear and well commented. Its modularity could be improved by isolating the game simulation from the graphical interface.

s2301773	12	Your program fails to detect many invalid spec. files, e.g., when multiples goals are assigned to the same piece, or when the entire grid is covered by pieces. Your game sometimes gets locked when trying illegal moves. The code would benefit from some commenting. It is not clear what are the distinct responsibilities of each classes (especially between Board and Grid). It also lacks modularity as the game simulation and the graphical interface are completely merged.
s2301811-s226162	13	Your program fails to detect clearly some invalid spec. files, either returning "null" as the error message, or simply crashing. On valid puzzles, it crashes when a movement would bring part of a piece out of the board. The program also does not stop after closing the window, and the window does not close at all after victory. You also forgot to change the color of the pieces which have reached their goal. The GameParser class need not to be instantiated or to have variables, you could have simply had one public class method that takes a String as input and returns a Board. Otherwise, the code is clear, well commented, and has good modularity.
s2301865	20	Your program detects most invalid spec. files and give a precise explanation. The game works as expected on valid puzzles. The code is clear, modular, and well commented.
s2301918-s2306710	13	Your program fails to detect some invalid spec. files, e.g., when goals are assigned to non-existing pieces, or when multiple goals are assigned to a single piece. The moves allowed by your game are not really consistent. Sometimes it is possible to jump multiple cells, sometimes we must move a piece one cell at a time, sometimes we can even move them diagonally. This can sometimes lead to accepting invalid moves. The window cannot be closed after victory. The method ReadFile.loadFile() should be a class method. The code is quite clear, although some commenting would have further improved its readability. It is quite modular, but you break the encapsulation principle by allowing the Board to give complete access to its pieces.
s2302009-s2302942	12	The archive does not match the required structured to be compiled with the command given in the statement. Your program fails to detect most invalid spec. files, e.g., when all the grid is covered by pieces. It works perfectly on valid puzzles. The code is clear and quite modular. If you override the *equals()* method, you must also override the *hashCode()* method accordingly. In Coordinates, it is a bit odd to make the height and width class variables, making them final instance variables would make more sense (otherwise it implicitly restricts greatly the use of this class, and could easily lead to bugs).

s2302195-s2304247	12	Your program fails to detect most invalid spec. files, and when it does it gives a very vague explanation. Your game mostly works as expected, although it sometimes crashes when the player clicks on a cell that does not contain a piece. The code is not modular, as the game simulation and the graphical display part are completely intertwined. If you override the *equals()* method, you must also override the *hashCode()* method accordingly.
s2302359-s2302054	9	The archive does not match the required structured to be compiled with the command given in the statement. Furthermore, your program is expected to take the name of the spec. file as an argument, not directly load "dataGame.txt" which is not even present in the archive. Your program only detects few invalid spec. files, and crashes on some illegal moves (when a pieces would have its top left part outside the board). Square pieces are also not always displayed as squares. The code is well commented. Some variables (in Piece are package private for no reason. Too much work is performed by the SlidingPuzzleGame class. Your code is not modular.
s2302371	14	Your classes should all belong to the right package w.r.t. the structure of your archive. Also, each file should be named by the class it describes (FileError). Your program fails to load valid 2x2 puzzles. The graphical interface and the reset/undo/redo buttons are a very nice addition, although the *redo* button sometimes lead to a crash because the scores somehow becomes -1. The code is quite clear and very well commented. However, it is not very modular, as the game simulation and the graphical interface are really intertwined. I also fail to see the point of the class *VariableArray*. Why not rely on Vector or ArrayList?
s2302437-s2300791	19	The game works well. The class SpecificationFileReader does not really need to be instantiated, and could simply contain a single public class method readPuzzle(). The code is clear, very well structured and modular.
s2302611	14	The archive does not match the required structured to be compiled with the command given in the statement. Your program fails to detect many invalid spec. files, e.g., when no goals are declared, or multiple goals are assigned to the same piece. The game works as expected on valid puzzles. Your code is clear and has some modularity, but there are some incoherences. GameBoard should not have two independent public methods *canMove()* and *movePiece()*. The latter should call the former before validating a move, to always ensure the piece is moved in a valid position. You were explicitly asked not to use **AWT**.

s2303125-s2303350	6	Your program detect precisely all invalid spec. files. Valid puzzles can be considered invalid because you swap *ncols* and *nrows* line 182 of SlidingPuzzle.java. Otherwise, the coloring of pieces that reached their goal which is quite erratic, and it is also possible to move a piece through other pieces (or jump over them). The code does not really respect the Object Oriented paradigm. All the work is done by the class SlidingPuzzle. It has no modularity. Dealing with the separate part of the project independently would have helped you notice that your program has many bugs, and helped you fixed them.
s2304029	9	Your program fails to detect some invalid spec. files, e.g., when multiple goals are assigned to a single piece, or when no goals are defined at all. Your game works okay, although it is possible to move some pieces diagonally, and make them overlap other pieces. This even makes some pieces somewhat disappear from the game. Although the code is well commented, its not modular at all. The parsing, the game simulation and the graphical interface are all combined together. Furthermore, you were explicitly required not to use **AWT**.
s2304076-s225089	12	The archive does not match the required structured to be compiled with the command given in the statement. Your program fails to detect some invalid spec. files, and when it does detect them, it stills open a graphical window which cannot be closed. The movements of the pieces are fine, but you do not change their color when they reach their goal, or inform the player of the win in a clear way (pieces simply cannot longer be moved). Furthermore, the window cannot be closed after victory. The code is clear, short and simple, with some modularity. You although declare variables like *Piece.winnigPiece* that you write on but never read. Try to avoid typos in the name of your methods and their first letter should be lowercase.
s2304210-s225753	13	Your program fails to detect many invalid spec. files, e.g., when multiples goals are assigned to the same piece, when the entire grid is covered by pieces, or when pieces overlap. On valid puzzles, your game crashes when the player either clicks on a cell where no piece is, or when it would bring a piece to overlap another one or to be partially outside of the grid. It also allows the pieces to be moved in any empty position, even if that involves going through other pieces. The code is quite clear and modular. You just forgot to deal with movements properly in the class Puzzle, which responsibility should be to ensure a movement is valid with respect to the game logic. You also use genericity for no reasons in your classes Piece and Goal (declaring Goal as *[public class Goal <Goal>]* does not do what you think it does). I think it is because you want to use them as a type parameter for generic ArrayLists, but that is not how it works.

s2304645-s2302095	16	The archive does not match the required structured to be compiled with the command given in the statement. Your program fails to detect some invalid spec. files, e.g., when multiple goals are assigned to a single piece, or when no goals are specified. It works well on valid puzzle, except that we expected that the pieces could be moved one cell at a time, and not necessarily pushed as far as possible in a given direction. The code is clear and modular.
s2304878	0	Incomplete project with 5 out of 7 files empty.
s2305323	16	Try to only give useful files in your archive. Your program detects invalid spec. files well and give a clear explanation. The game works well on valid puzzles. The numbering and the goal visualization are nice features. Your classes should be declared in the package be.uliege.montefiore.oop to match your archive structure. The code is very well documented. It is however a bit complex, and it is not always clear to understand the precise responsibility of each class. For instance, the class GameState should be more independent, and should not need to rely on ConfigurationChecker (which really depends on the way we defined our spec. files) to check whether or not adding a new piece breaks the game.
s2305392	15	Your program fails to detect many invalid spec. files, and when it does, the explanation is not precise. On valid puzzles, your game works well, although square pieces are not always displayed as squares. The code could use some commenting. It has some modularity, although it should be the PuzzleInterface responsibility to deal with RGB colors, and not the Tiles. The class Game could have been made simpler by making the methods static. There is no reason to instantiate this class since the instance variables you declare are only used in one method (*Game.read()*), and there is no point in storing them after exiting this method.
s2403896	15	The archive does not match the required structured to be compiled with the command given in the statement. Your program fails to clearly detect invalid spec. files, e.g., when no goals are declared, or when multiple goals are assigned to a single piece. The game works perfectly on valid puzzles, except that square pieces are not always displayed as squares. The code is clear, and quite modular. There is however some visibility issues: the class variables in ReadGameSpecFile are public but you also have public getters, and more importantly, the coordinates of the PuzzlePiece are also public.

s2405523	12	The archive does not match the required structured to be compiled with the command given in the statement. Your program fails to detect some invalid spec. files, and when it does detect them, it sometimes still open a window and then crashes, e.g., when a goal is assigned to a non-existing piece. Otherwise, the game works perfectly on valid puzzles. The code is clear and well commented. It is however not modular, as the game simulation is completely intertwined with the graphical interface. When you define a class, make sure its methods cannot corrupt the state of an instance. In particular, in the class *Zone* , the length and width of the area cannot be changed after instantiation, but one can change the position of the bottom right corner, which then makes the internal state of the Zone incoherent.
s2405658-s203252	18	The program works as expected, except that it fails to detect a few invalid spec. files. The code is clear, well commented, and is quite modular. There are a few details that can be improved. The variable *Goal.ngoals* should have private visibility, and it should not be the responsibility of the Pieces to deal with RGB colors.
s2406888-s214756	14	The archive does not match the required structured to be compiled with the command given in the statement. Your program fails to detect some invalid spec. files, e.g., when multiple goals are assigned to a single piece. Otherwise, the game works perfectly on valid puzzles. The code is quite clear, although it would have been better to clearly separate the spec. file parsing part, from the GUI management. The puzzle simulation is well independent from the other parts. The class Piece has some redundant methods, *translate()* is enough, no need for *SetNewXpos()* and *SetNewYpos()* (which should start with lowercase letters). Also, these methods name are not coherent with what they do: they do no change the value of x/y for the argument, but translate them.
s2409266	16	Your program fails to detect some invalid spec. files, e.g., when the pieces cover the whole grid, or when a goal is assigned to a non-existing piece. Otherwise, the game works perfectly on valid puzzles. The code is clear and well commented. It is not really modular as the game simulation is intertwined with the graphical interface. Be careful with the public method your provide. In *Grid* , it should be the responsibility of the class to deal with the case when one tries to add too many pieces (via *addPiece()*). But in your code, the property that no more than *maxPieces* pieces are added must be enforced from the outside.

s2409345	9	The archive does not match the required structured to be compiled with the command given in the statement. You were asked to make *SlidingPuzzle* the main class of your project, but this class does not have a main method, and two main methods are declared in two other classes. Your program fails to detect some invalid spec. files, e.g., when a goal is assigned to a non-existing piece, or when multiple goals are assigned to a single piece. The game works fine on valid puzzles, except that square pieces are not always displayed as squares. Some methods in your code are too complex, especially in the class *SlidingPuzzle* . Try to decompose it into smaller parts to improve its readability, and try to decouple the game simulation from the spec. file parsing and the graphical interface. There is no reason for the *Color* variables to have public visibility, and the *Goal* variables to have package private visibility.
s2409461	16	Your program detects most invalid spec. files. The game works well on valid puzzles, except that it does not stop cleanly if the window is closed midgame. The code is quite clear although some things could be made simpler. For instance, either give a more explicit name to the exception class *Except* , or use the class Exception directly. In *Piece.isConflictFinal()* , you should deal with the case where at least one piece does not have a goal, and not delegate that responsibility away. Be careful that if you give access to every pieces on the Board freely (via *Board.getPieces()*) then it is possible to move the pieces freely from the outside without enforcing that the moves are lawful.
s2409481	14	Your program detects most invalid spec. files and give a precise explanation (the only issue occurred when no goal were defined at all). On valid puzzles, it is possible to move a piece through other pieces, and to perform diagonal moves that should not be allowed. The code is clear and modular. There are however some problems in the class Piece. The instance variable *Piece.cells* is not very useful and can be very misleading. You only use it when you initialize the board (so before any move is performed), but you never update it again, although a public method allows to access it. Therefore, after a move, this variable is no longer coherent with the Piece current position.