Technical Plan - Triangle Puzzle

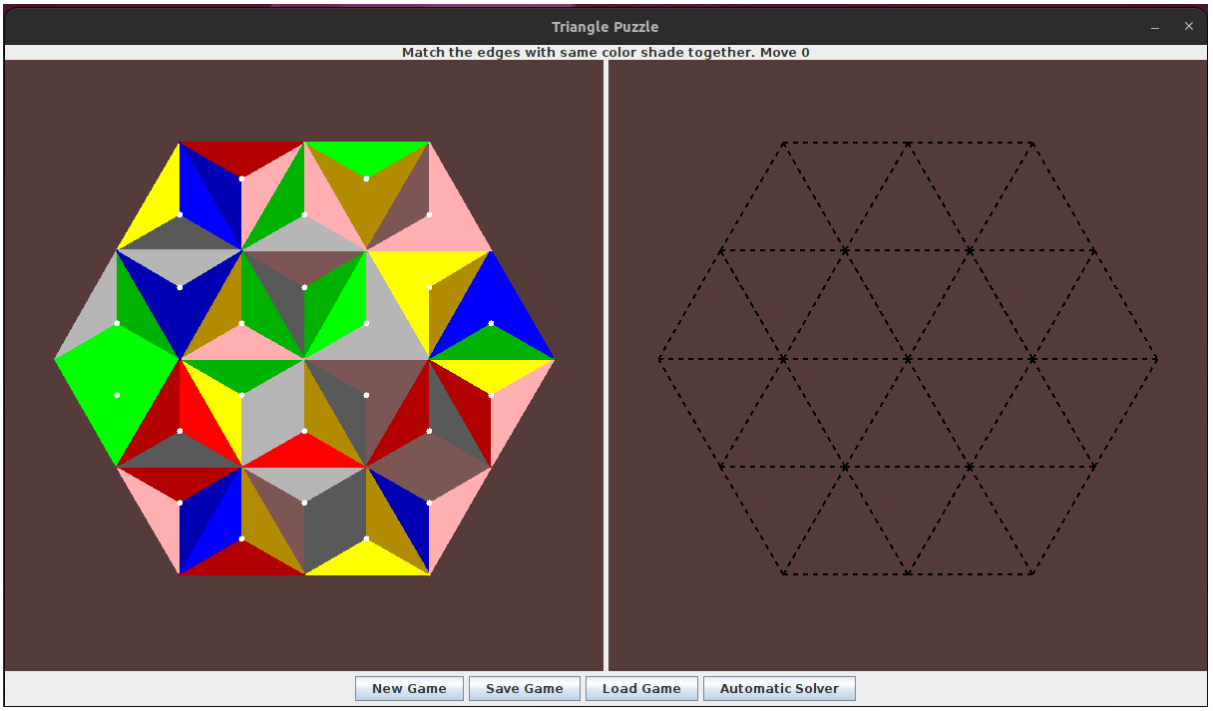
1. Personal Information

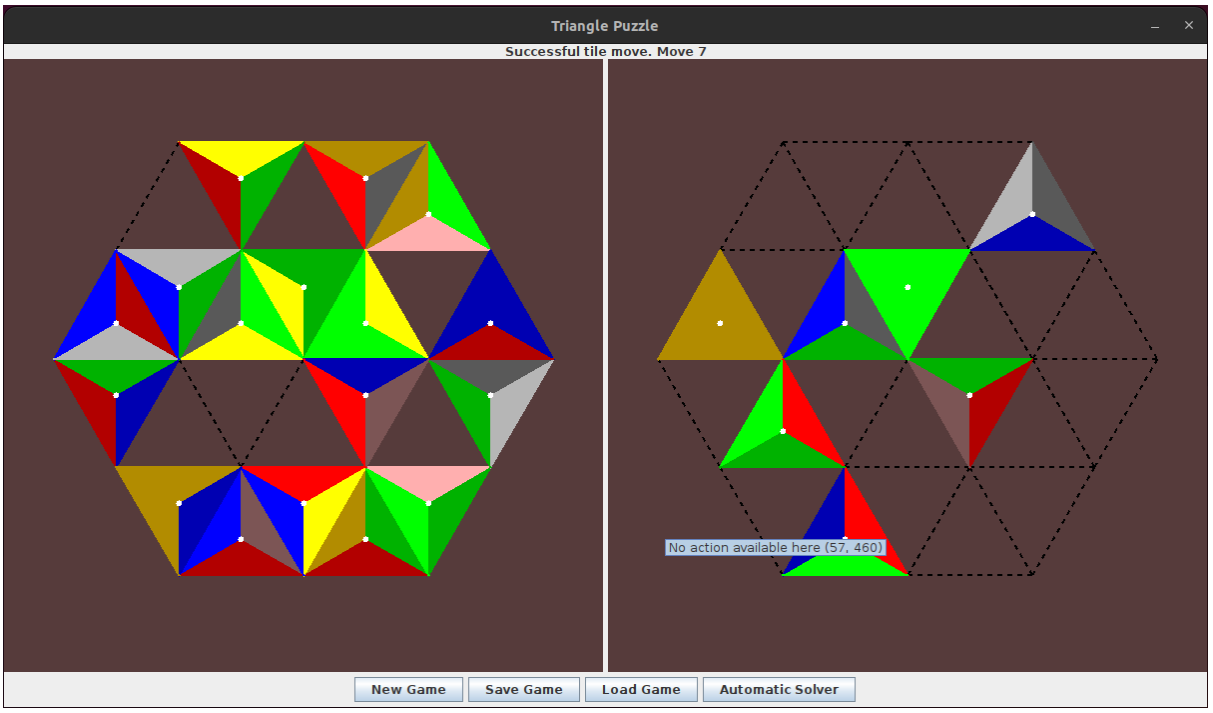
- Title: Triangle Puzzle (103)
- Name: Nguyen Duc Tam
- Student Number: 906968
- Study Program: Bachelor of Data Science
- Study Year: 2021-2024
- Date: 27.04.2023

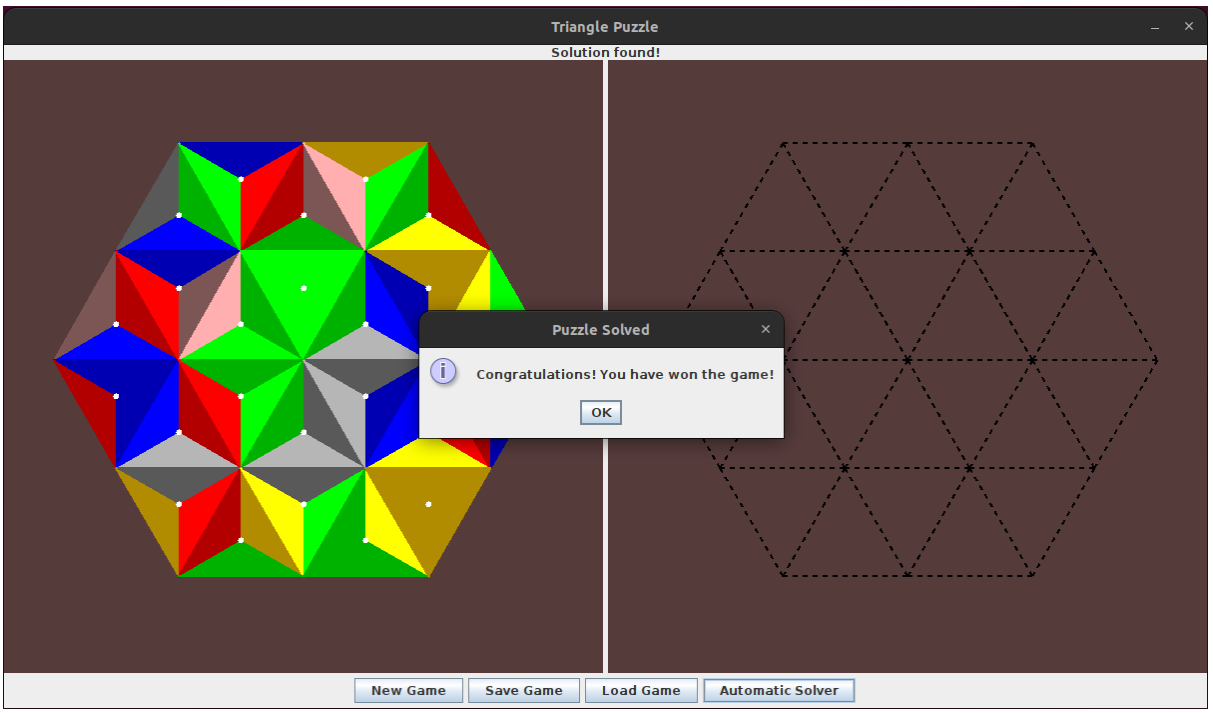
2. General description and difficulty level

- Project Triangle Puzzle is about creating a puzzle game consisting of triangles. To be more precise, the puzzle is much like the Tarsia puzzle. In this puzzle, the pieces will be set on a hexagonal area so that for every triangle the symbols on the corresponding edges match with the symbols of the adjacent triangles.
- **Level of difficulty: Demanding**
- In the graphical interface, the player needs to arrange the triangle tiles so that all the adjacent edges have matching color shades (there are 6 different colors, each with 2 shades of color: light and dark).
- The program can generate a random and valid solution. Then it shuffles and rotates the triangles to start a new game for the player.
- The player can use the mouse to browse through all the triangles. There is a small tooltip that goes with the mouse to tell the available actions.
- The player can use the mouse to put on and remove the triangle tiles from the game board by drag-and-drop mouse movement. The game has an empty board on the right to show all the tiles that are removed from the game board.
- The player can rotate the triangles by right-clicking the mouse on the tile.
- The program automatically detects when the puzzle is solved. In such a case, it pops up a dialog with a notification message.
- The program has saving and loading game functionality.
- The program has a solver to automatically solve the game if the player wants.

3. User Interface

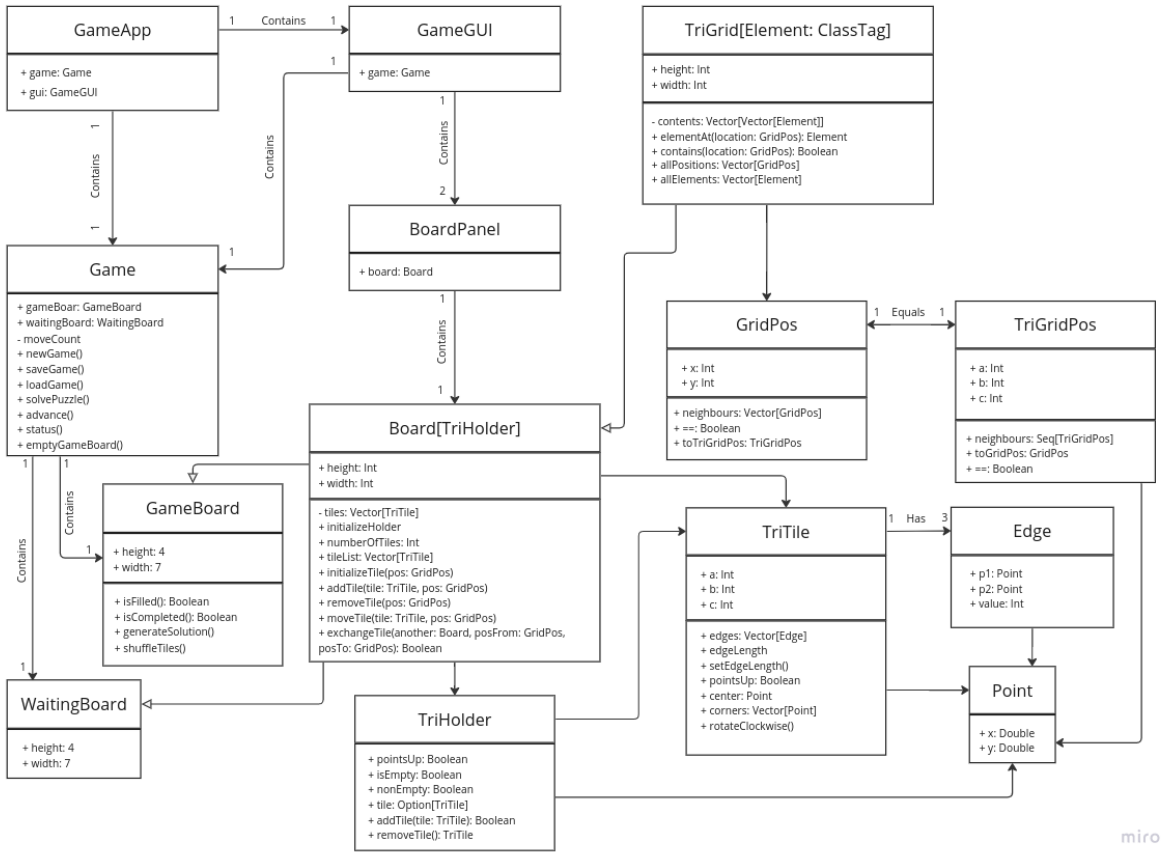






- To launch the program, the user needs to run the file `project-triangle-puzzle/src/main/scala/gui/GameApp.scala`.
- The boards on the left hand side and right hand side are game board and waiting board, respectively. If all the tiles on the left board is arrange such that all the edges have matching color shades (e.g., light-green adjacent to dark-green in the middle of the second row), the player wins the game.
- The player can use the mouse to drag the tile from one board to another and vice versa, and can use the mouse to right-click on the tile to rotate it clockwise.
- Other than that, the program has 4 buttons at the bottom of the window. The button names are very descriptive in terms of what they do.
- **Note:** in the technical plan, there are next and previous buttons that allow users to visit their previous moves. This functionality **is not implemented** eventually.
- **Note:** when the player clicks the button `Automatic Solver`, the program is not able to show the moves of the solving process on the GUI. It will freeze the program a little bit while solving and only show the final result.

4. Program Structure



The structure of the program is divided into two main parts:

- engine: this folder contains all the classes that handles the logic of the game (e.g., triangle tiles, board that contain tiles, edges that contain the value corresponding to a triangle tile, etc)
    - engine.grid: this folder contains the classes that work as interfaces to convert between triangular coordinate system and regular Cartesian coordinate system. Personally, figuring out the math and logic of the classes in this folder is the hardest part of this project. Once the logic of this grid system is figured out, it builds a strong foundation for other parts of the program.
- gui: create the graphical user interface to the program, where the player can use a mouse to easily interact and play the game.

5. Algorithm

- Algorithm to generate a random and valid solution: in UML diagram, function `generateSolution()` is implemented in class `GameBoard`. Pseudocode:
    - generate a random triangular tile on the leftmost upmost position in the hexagon
    - the tile on the right of that first tile has matching edge with the first tile
    - other unshared edges of the second tile can have arbitrary value

    - repeat the process for other tiles in the hexagon
- Algorithm to shuffle and validly rotate the triangles in the hexagon before starting the game: method `shuffleTiles()` in class `GameBoard`. Pseudocode:
    - randomly choose 2 tiles from `GameBoard` and swap them
    - repeat the process 24 times
- Algorithm to detect when the puzzle is solved: method `isCompleted()` in class `GameBoard`. Simply iterate through the tiles and check that all neighbors of the tiles sharing the same edges have the same value on that edge.
- Algorithm to automatically solve the puzzle: the game can be considered a Edge-Matching problem [6, 7]. The solver in this program uses a backtracking algorithm to recursively trial-and-error find the solution [7, 9, 10]. Initially, I have attempted some complex, high-performance algorithms such as A*, satisfiability (SAT) solver [8], but these algorithms only find the end result, and not record the whole process. It is also in general more difficult to implement the algorithms above compared to backtracking.

6. Data Structure

- The program does not use some custom data structure, and it only uses Scala built-in data structures.
- Two most important data structures that the program uses extensively are list and hashmap (i.e., different collections like List, Vector, ArrayBuffer, Seq, and Map in Scala).

7. Files

To be able to save multiple information of a game state (e.g., how many tiles are on/off board, locations of tiles on/off boards), a simple file structure such as CSV will not be a good choice for me. Therefore, I choose JSON files, and an example of a file can be as follow:


```
{
 "moveCount" : 0,
 "gameBoard" : [
  {
   "a" : -1,
   "b" : 2,
   "c" : 1,
   "edges" : [ 4, 22, 1 ]
  }
 ],
 "waitingBoard" : [
  {
   "a" : -1,
   "b" : 2,
```

```
    "c" : 0,
    "edges" : [ 44, 66, 2 ]
   }
 ]
}
```


8. Testing

The testing of the program can be divided into two parts:

- engine (backend) test: almost all the classes in folder engineer have their own unittest. A large number of unittest have been created and put inside /src/test/scala, and these test files play a crucial role in the operation of the program.
- GUI test: the GUI was tested mainly using the graphical interface of the app, and using mouse tooltip to gather information about tiles, boards, etc.

9. Known bugs and missing features

- Because of the `generateSolution()` algorithm, there is a chance that some tiles cannot have any valid edge value due to randomness in previous tiles. Therefore, instead of 4 variables (a,b,c,d) as in the project example, the program has 6 variables (red, green, blue, yellow, pink, gray). Since I increased the number of variables to 6, I have not experienced a case where a valid solution is not able to be generated. However, according to probability, there is still a slight chance that a new game cannot be created.
- Right-clicking the mouse will rotate the triangle tile clockwise. However, after the right click, the user needs to move the mouse a little bit so that the change appears in GUI. Otherwise, the GUI still shows the same tile.
- For once, when clicking some of the four buttons at the bottom, and doing something in the popup dialog, the program prints out some exception error in the terminal, but the GUI still runs normally. However, I cannot reproduce the error so I cannot describe it further.
- After clicking the Automatic Solver button, ideally, the program shows the steps of the solving process. Even better, it allows the player to play after showing some of the solving steps. However, now, when the user clicks the Automatic Solver button, the program will freeze, and it will only show the final solution after the solving process is done. I have investigated, and the problem of the GUI cannot be updated during the solving process is because of the recursive calls of the solving function and the Thread in Scala Swing. With more knowledge about threading and concurrency, I think I can take advantage of scala.concurrent and solve this issue.

10. 3 best sides and 3 weaknesses

Best sides:

1. The math behind calculations in the triangular coordinate system has been tested carefully, and it works very well. These formulas and logic can be used in different applications, and increases the scalability of the project.
2. The code is well-documented, and is clearly separated in different classes and packages. The functions are divided as per their classes, and help reduce the

redundant codes. It also helps with the scalability and maintainability aspect of the project.
3. The program has many test files (almost all methods in classes in the engine are covered). The tests ensure that the logic of the game works fine, and it helps a lot when the engine is integrated with the GUI, especially in the debugging process.

Weakness:

1. The conversion between triangular coordinate system and regular Cartersian coordinate system is handled by `GridPos` and `TriGridPos`, and predefined hashmap in engine.grid. This manual conversion restricted the size and shape of boards in the game to be exactly 24 tiles with 4 rows, and the number of tiles are 5, 7, 7, 5, respectively. It makes the program have a static shape and size, and the project cannot be scaled up.
2. The GUI uses color-matching for edge-matching of the tiles. In my opinion, the GUI is not very aesthetic, and can even annoy the eyes of the players. Better and more aesthetic graphical design will definitely make the project look a lot better.
3. The solving algorithm runtime is not consistent. The GUI cannot show step by step moves in the solving process.

11. Deviations from the plan, realized process and schedule

Tell here in general terms about the order in which the project was finally implemented, for example, in steps of two weeks each. Where did the process deviate from the initial plan and why? How well did the time estimate in your plan match with the reality? What were the biggest differences? What about the order in which you chose to implement the different parts of the project? What did you learn during the process?

20.02.2023: submit technical plan
24.02.2023: finish the course rounds, starting project.
25.02.2023 - 10.03.2023: familiarizing with Swing and trying to create a dummy GUI
11.03.2023 - 15.04.2023: finish backend
15.04.2023 - 23.04.2023: implement GUI
24.04.2023 - 26.04.2023: implement solver

12. Final evaluation

Conclusion of the self-assessment, where you can summarize some of the above things.

Evaluate the quality of the final program, discuss its good and weak aspects. Are there significant shortcomings and where do they emerge (a good explanation in the document can replace the small shortcomings)? How could the program be improved in the future? Could the choice of solution methods, data structures, or class structure be better? Consider how the structure of the program is suitable for making changes or extensions?

Finally, evaluate what you would do differently if you started the project again from the beginning.

- Overall quality: good; the program has all the requirements of the project in the demanding level. The final program runs smoothly and does not show a lot of bugs from my personal experience.
- Significant shortcoming: the static hashmap described in 3 weaknesses section makes the program unscalable.
- The program can be improved using a better strategy to convert between GridPos and TriGridPos, and enhancing GUI.
- In my opinion, my code for the program is quite neat and clean. The structure is clearly divided into sub-sections, and the methods are divided into different classes. I find it quite easy to follow up with my code after a while, and changes or new methods can be added to the program quite easily thanks to that neat structure.
- I also learn more about OOP thanks to Scala explicit and detailed functions regarding the OOP aspect.

13. Reference and links

[1] K. Abuhmaidan, M. Aldwairi, B. Nagy, “Vector Arithmetic in the Triangular Grid,” Entropy (Basel), Mar 2021. Accessed: Mar 28, 2023. doi:10.3390/e23030373. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8004019/

[2] Boris, "Triangle Grids." BorisTheBrave.Com. https://www.boristhebrave.com/2021/05/23/triangle-grids/ (Accessed: Mar 25, 2023)

[3] Boris, "Implementation of triangular grid in Python." github.com. https://github.com/BorisTheBrave/grids/blob/main/src/updown_tri.py (Accesse: Mar 25, 2023)

[4] Aalo University Programming 1 (O1) course. Module RobotWorld

[5] Aalo University Programming 1 (O1) course. Module o1.grid

[6] “What's the theory behind this puzzle?” stackoverflow.com. https://stackoverflow.com/questions/39125810/whats-the-theory-behind-this-puzzle (accessed April 22, 2023).

[7] “Edge-matching puzzle” wikipedia.com. https://en.wikipedia.org/wiki/Edge-matching_puzzle (accessed Mar 31, 2023).

[8] Marijn J.H. Heule, “Solving edge-matching problems with satisfiability solvers,” [Online]. Available: http://www.cs.cmu.edu/~mheule/publications/eternity.pdf (accessed April 22, 2023).

[9] “Solving edge-match puzzles with Arc and backtracking” righto.com. http://www.righto.com/2010/12/solving-edge-match-puzzles-with-arc-and.html (accessed April 25, 2023).

[10] “Sudoku solver with backtracking in Python” github.com. https://github.com/techwithtim/Sudoku-GUI-Solver/blob/master/solver.py (accessed April 26, 2023).

[11] “Computer graphic with 50 triangles in Scala” github.com. https://github.com/philipschwarz/computer-graphics-50-triangles-scala/blob/master/src/main/scala/TrianglesPanel.scala (accessed Mar 15, 2023).

[12] “Lesson: Using Swing Components” docs.oracle.com. https://docs.oracle.com/javase/tutorial/uiswing/components/index.html (accessed April 15, 2023)

[13] “GUI Programming” otfried.org. https://otfried.org/scala/gui.html (accessed April 15, 2023)..