# Escape Game Design Document

Andrew Whitney

12 December 2020

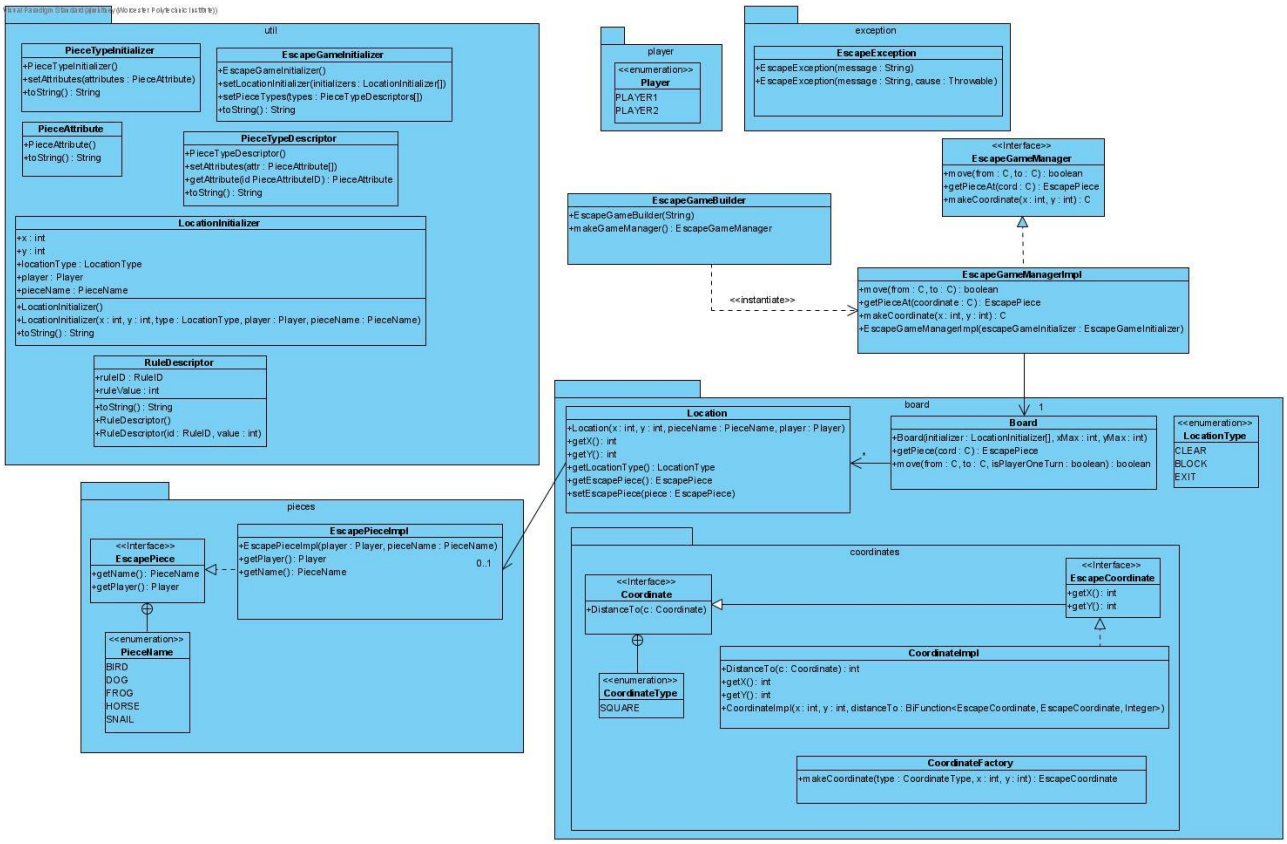
# Alpha

First thing done was read over the requirements for the alpha release, taking notes of the key parts in the Blog.md document. Next was taking note of Table 5 in the Development Document which would help develop the move() function which was a major part of the Alpha release. After doing this, I got caught up on the course modules that would help with this project and watched all office hours and attended supplemental lectures.

Since the document stated for Beta more coordinate types would be added other than SQUARE, implemented a factory method pattern for creating coordinates seemed like a good choice. Implemented only SQUARE, so other types would throw an error. Brainstormed where to place the locations and pieces. Initial thought was either a 2D array or hash map. Decided to separate this from the manager and created a Board class which would store the HashMap. Made a note to investigate Dependency Injection but was currently just creating the board in the manager's constructor. Created a Location class which would go inside of the Board's HashMap and store each location's information. I made these design choices in a attempt the follow the Single Responsibility Principle. After creating some helper methods, and some getters/setters getPieceAt() was working along with makeCoordinate(). An adjustment was needed to makeCoordinate() but due to how the project was set up the change was easy and did not cause any other tests to fail that had previously passed regarding other requirements. After running a code coverage test on my code, I added some more tests to address some areas missed.

When getting started on move(), I reread the Developers Guide and made not of which moves returned true and which returned false. I made a note of having the manager pass the information needed to make the entire move in the board then just return true/false to the manger after the move is done. I decided to do this to keep the information encapsulated.

I had to modify one tests and some of the code after learning moving a player's piece to the same location returned true in alpha. The change was simple since I was following design principles. After some refactoring, including moving unused files into packages I felt appropriate, and adding Javadoc comments on some private methods, the Alpha release was ready to be submitted. Below is my Alpha Submission as a UML Diagram:




# Beta

First thing I did was take note of the requirements of Beta and look for any changes from Alpha to Beta. I noticed the CoordinateFactory would now need to be able to create TRIANGLE Coordinates. TRIANGLE coordinates would need a different way of testing DistanceTo, so I moved the distanceTo Lambda from Alpha into a DistanceCalculate class in the coordinates package. I then added a way to calculate tirangleDistanceTo. Refactoring this way helped make the code more readable and did not give the CoordinateFactory more than one responsibility.

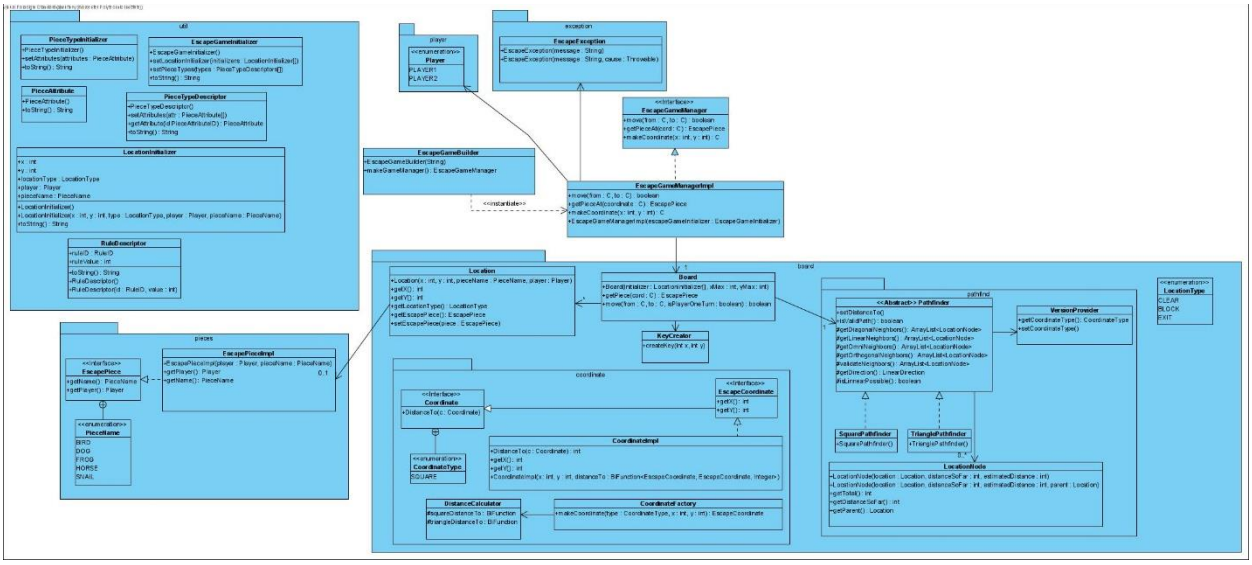
I must modify some of the code in the Board class to support infinite boards, this did not take long, and when I reran my tests from alpha, none of them broke.

Since EscapePieces now held more information, it was necessary to come up with a better way to create them. They currently were being created in the constructor of the Location. I created a EscapePieceFactory that would be initialized with the PieceDescriptors provided by

the Builder. This broke up my code and made it a lot more readable and organized. Any changes now to the creation of the pieces was in one location instead of in the same location changes to the Location class were made.

To Make pathfinding on the board clear, clean and easy to modify/read, I utilized the template pattern for Pathfind and created a SquarePathfind and a TrianglePathfind. I created its own package so I could then place helper classes such as LocationNode in there and not expose them to the rest of the board package.

I removed unused code and placed it in the blog, such as the unblock code since it was no longer a requirement for the Beta. I also made KeyCreator protected since it will only be used by the board package. Added comments as needed and submitted my project.



# Final

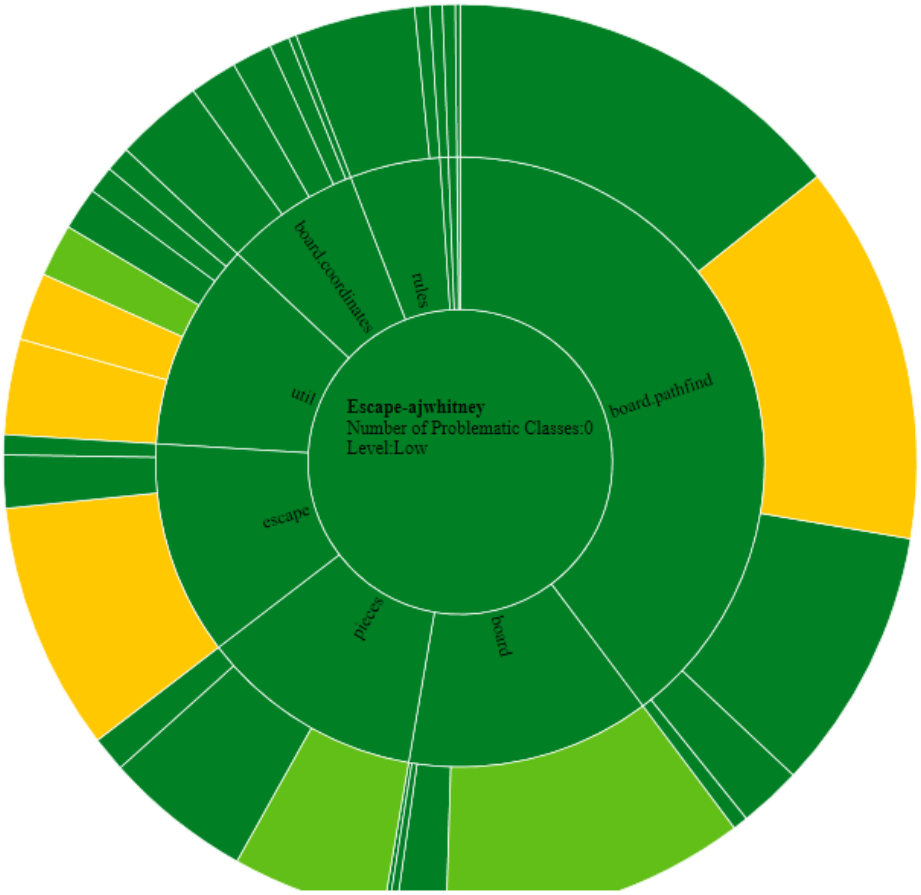
For the final implementation there are not many changes. First is added the game rules, implementing BLOCK location types of SQUARE boards and LINEAR movement patterns. Also implementing the observer function. These changes did not require refactoring of previous work, so it was simple to implement. The EscapeGameManager communicated with the observers so an array of the added observers was stored there. move() was the only function that would communicate with the observers. I utilized the EscapeException to notify the observers. I would catch all exceptions thrown and notify observers of those exceptions. I created a

RuleHandler class so the Manager does not take on that responsibility and would check with it to see if win conditions have been met.  Ensuring tests for the observer and rule conditions pass, along with fixing old tests that fail due to new constraints, the Final was now finished.

# Analysis

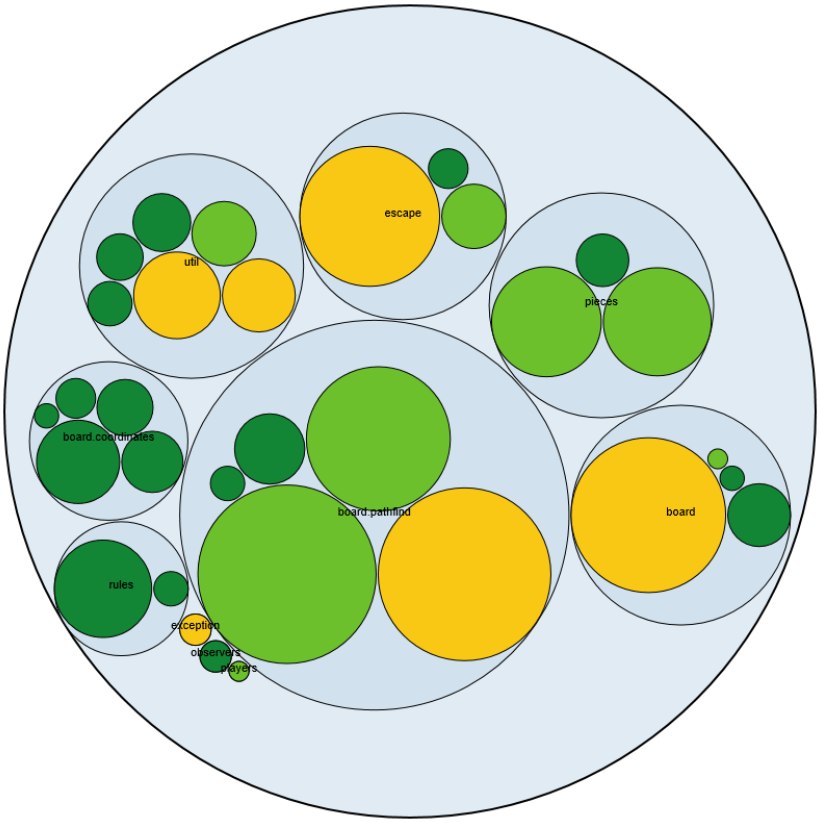
I utilized CodeMR to look at different OO metrics of my code and see how well I designed my program.

## Lack of Cohesion




When looking at lack of cohesion, I was pleased to see all my packages had a low lack of cohesion.  My Pathfind class had a higher than desirable lack of cohesion, however along with EscapeGameManagerImpl.

C3 (Coupling, Cohesion, Complexity)




Taking a look at C3, I was overall pleased with the results. Three of the bigger classses had more exposure, and if I were to continue developing this product, I would take a look at how to improve the desgin of those classes and reduce exposure. If I were to do this project over again, I would try to experiment more with Dependency Injection, and see if it improved my scores. I would also look into break up Pathfind and Board more to see if I could lower its score.

# Reflection

Never having designed a project while applying OOAD Principles, planning was the hardest, yet most useful tool through this project. Reflecting on class modules, looking at examples of Design Patterns, and writing down a plan of the process was fundamental in keeping the project organized and reduce clutter. Refactoring once these ideas were in code was essential to keeping readable, clean code. Pairing design patterns with packages helped organize these files and keep the project understandable. This was evident when changes had to be made from release to release. Parts of the project implemented in a clean, thought out manner were easier to modify than parts placed in where it was convenient. Test Driven Development (TDD) sped up the process of writing effective code. Able to write tests that match the requirements for the release, less time was spent on re-reading the developer's guide for requirements and edge cases and that time was spent going over tests written down in the TODO.txt in the project directory and modifying the code to pass the tests. I enjoyed TDD and if I were to work a big project with a lot of requirements, I could see myself using it again. On smaller projects it feels less practical, but real-world application of these principles seems to be better than trying to program without tests. I have learned a lot from this course and feel a lot more confident in my programming abilities and understanding of how to design good software.