Object Oriented Implementation of AI Chinese Chess Game

(Project 6 Report, Final Report)

Team Members:

Keyu Wu

Puyan Zhang

Status Summery:

title of the project:

Object Oriented Implementation of AI Chinese Chess Game

team members:

Team name: *bakamaka-lalala*

Keyu Wu:

*S*tudent ID: 107114346

Email: **kewu6458@colorado.edu**

Tel: 6265416005

Puyuan Zhang:

Student ID: 107207898

Final State of System Statement:

All major goals have been accomplished.

There is an UI implemented using thinner.canvas that could detect mouse click the return the clicked position. We transfer that to index on the board and using the index to do the selection of chess pieces. when you click on a piece, a blue indicator would show up to in indicate which position you clicked on and several black indicators would show up to indicate all the possible moves. You may then click on any grey indicator position to move the piece you selected. You may also switch selections by clicking other pieces.

One basic Artificial Intelligence algorithm is also implemented. The algorithm is using alpha-beta pruning method which is believed to be much faster the simple min-max algorithm, but it still runs super slow. A search depth of 4 is used in the early game, 6 is used in later game.

The game is playable just one need to be very patient, it usually take seconds to do a depth 4 search and minutes to do a depth 6 search. I am also considering multithreading since python only using one thread by default. I tried to implement it but using multithreading but I found it is too hard to do because the alpha-beta pruning method is an recursive method that stores two critical data while going recursing, to implement multithreading I would need to implement communication between threads which would be a great amount of work. I am also considering implement dynamic programming which boost the algorithm by storing the intermediate results, due to time constraint, I did not implement that either.

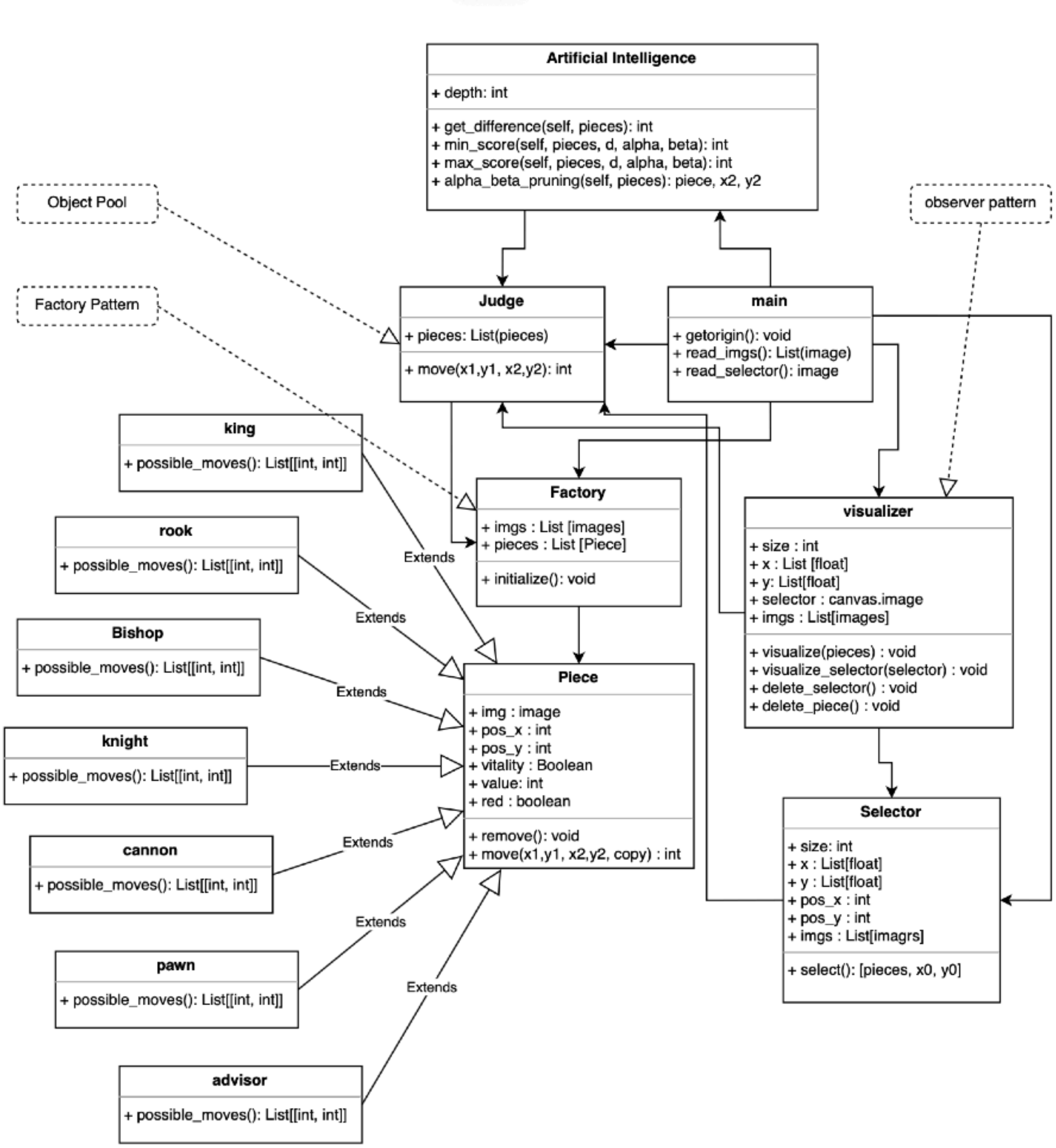
Changes:

What changed compare to Project 5?

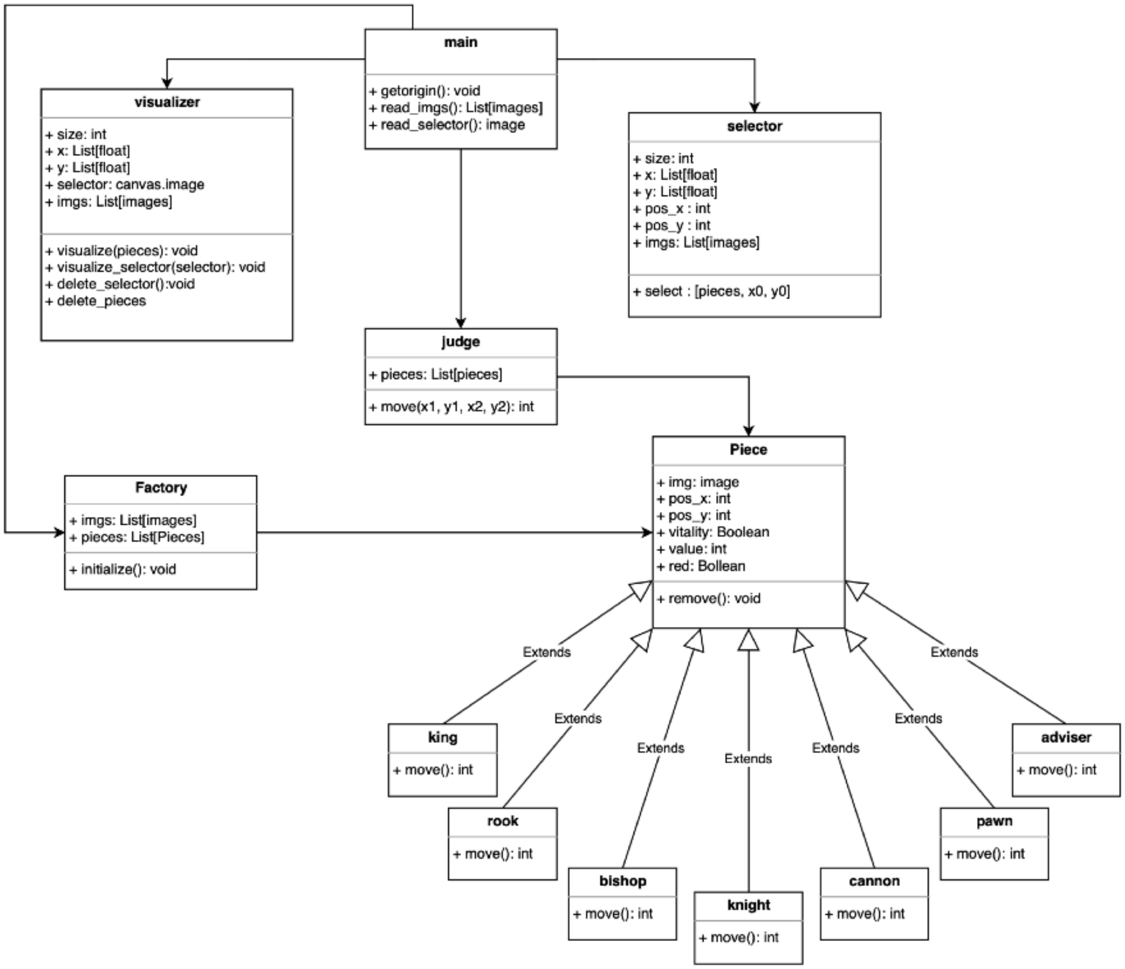
By the time of Project 5, we already completed the UI. What is new in this report is an AI, the implementation of the rules(which piece could go which where) and a few modifications.

Class Diagrams:

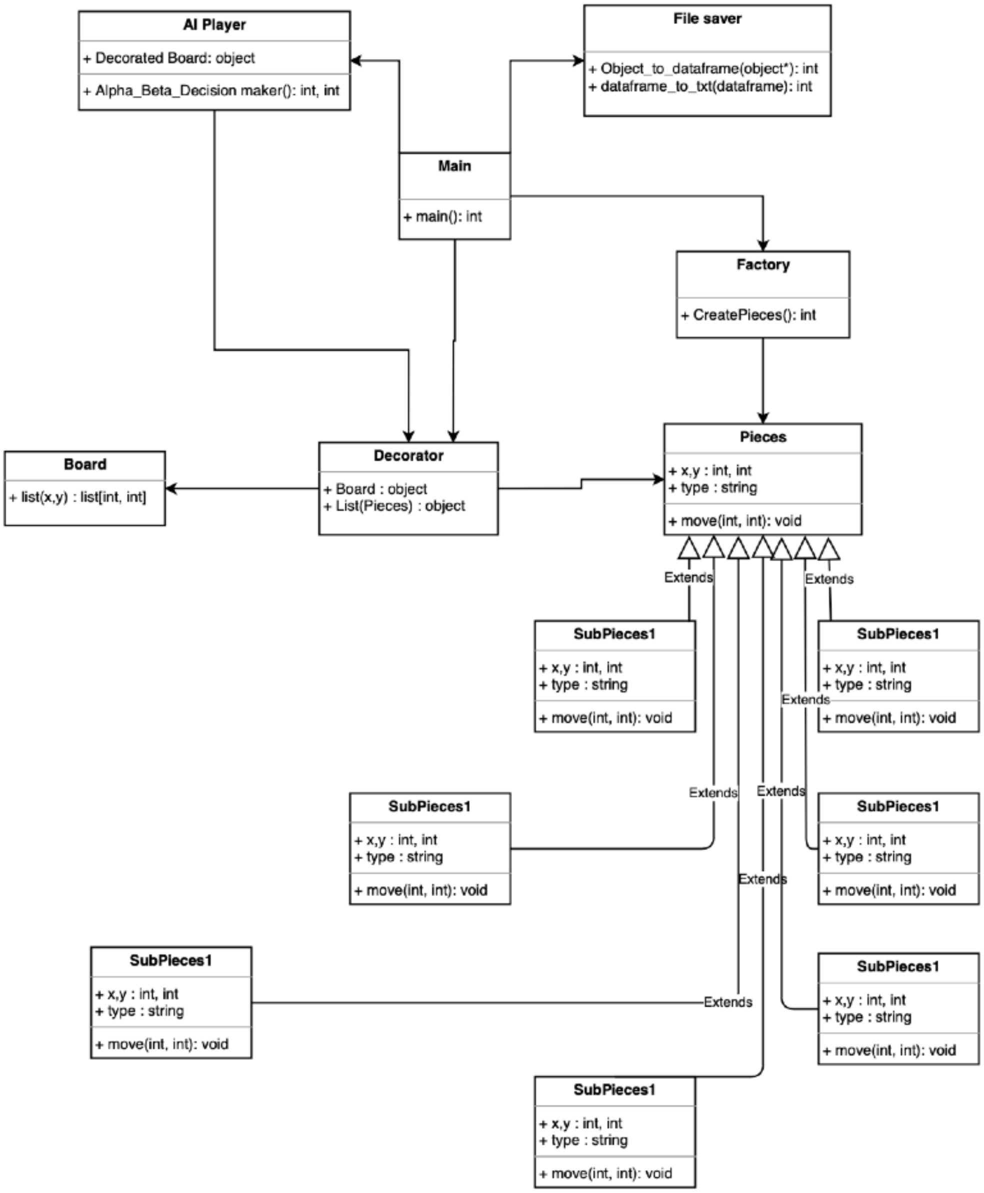
Final Class Diagram as of project 6:


Artificial Intelligence
+ depth: int
+ get_difference(self, pieces): int
+ min_score(self, pieces, d, alpha, beta): int
+ max_score(self, pieces, d, alpha, beta): int
+ alpha_beta_pruning(self, pieces): piece, x2, y2
Object Pool
observer pattern
Factory Pattern
Judge
+ pieces: List(pieces)
+ move(x1,y1, x2,y2): int
main
+ getorigin(): void
+ read_imgs(): List(image)
+ read_selector(): image
king
+ possible_moves(): List[[int, int]]
rook
+ possible_moves(): List[[int, int]]
Bishop
+ possible_moves(): List[[int, int]]
knight
+ possible_moves(): List[[int, int]]
cannon
+ possible_moves(): List[[int, int]]
pawn
+ possible_moves(): List[[int, int]]
advisor
+ possible_moves(): List[[int, int]]
Extends
Extends
Extends
Extends
Extends
Extends
Extends
Factory
+ imgs : List [images]
+ pieces : List [Piece]
+ initialize(): void
Piece
+ img : image
+ pos_x : int
+ pos_y : int
+ vitality : Boolean
+ value: int
+ red : boolean
+ remove(): void
+ move(x1,y1, x2,y2, copy) : int
visualizer
+ size : int
+ x : List [float]
+ y: List[float]
+ selector : canvas.image
+ imgs : List[images]
+ visualize(pieces) : void
+ visualize_selector(selector) : void
+ delete_selector() : void
+ delete_piece() : void
Selector
+ size: int
+ x : List[float]
+ y : List[float]
+ pos_x : int
+ pos_y : int
+ imgs : List[imagrs]
+ select(): [pieces, x0, y0]

As of Project 5:



Original Thought as of project 4:



UML changes explanation:

As you may observed from the diagram above, there are no major changes on the initial thoughts. It is just when you are into actual implementation, you unconsciously choose the structure that are more convenient. From the initial thought, I decide to create a board object; but in actual implementation, I find tkinter.canvas is a more convenient library to use. I also abandoned the decorator method since the board object itself does not exist.

Third-Party Resources:

Reference of the images: Chinese Chess Wizard, https://www.xqbase.com

OOAD Process:

OOAD did offer a good sample structure via those design patterns. which makes the design of our project easier.

Also, the idea that make everything into objects is logically friendly. Objects are always less abstract than other sort of code, which makes it more easier to think through.

But, the idea of making everything into objects limited the performance of my Artificial Intelligence greatly. My AI keeps creating and deleting tons of object that consumed much of the performance.

Project Repo:

https://github.com/MiracleWKY/OOCC