電腦對局導論期末報告

姓名：顏百謙 系級：資工碩二107級 學號：R07922135

編譯方式

- 在R07922135/code/ 資料夾底下
- make local
- 執行檔名為R07922135

System overview

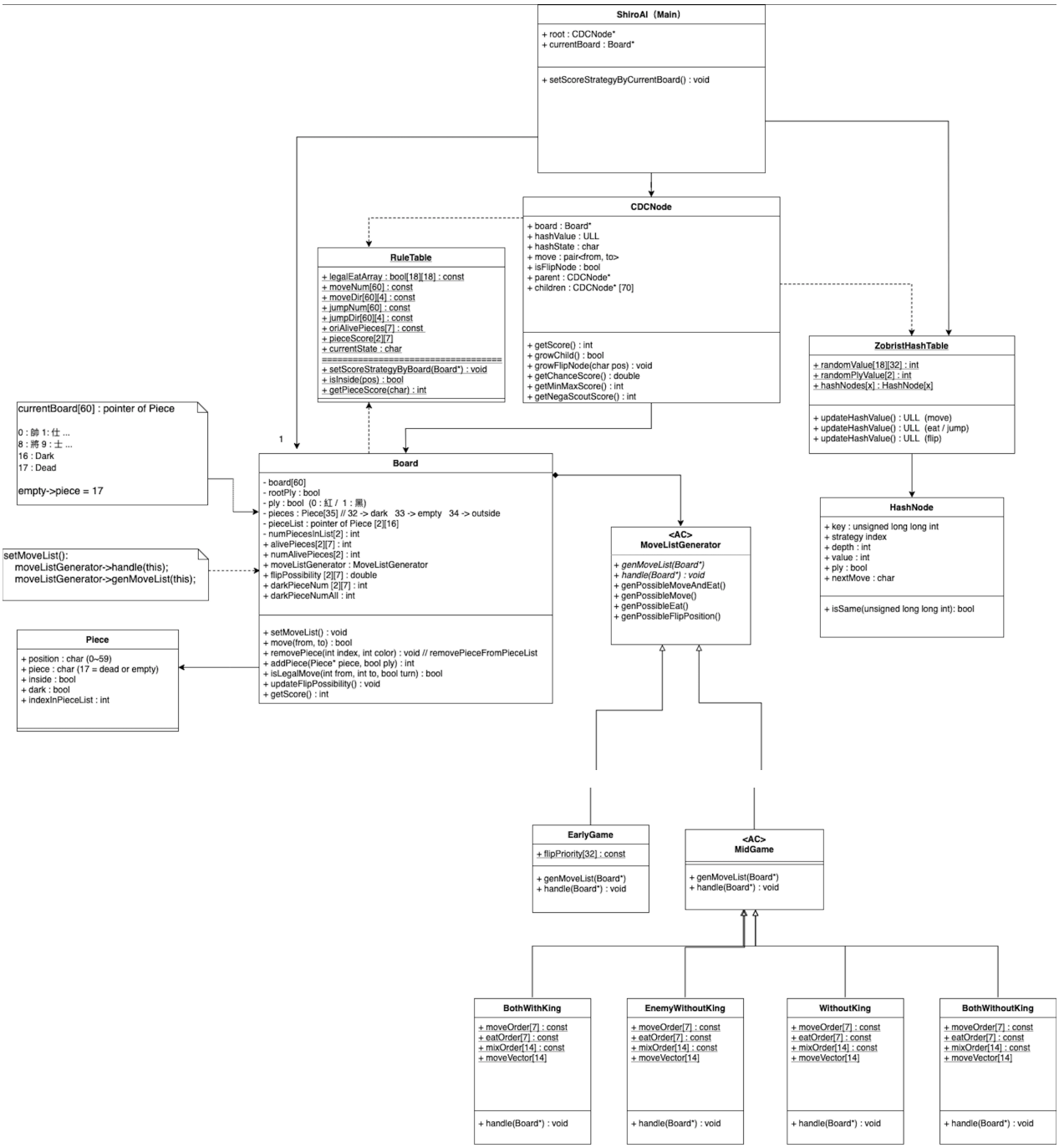
期末的Project是搭配NegaScout演算法，Transposition Table with Zobrist hash，以及走步排序等等技術實作暗棋AI，並與修課同學進行對打決定名次。

在此次的Project中我們可以學到如何實作教科書中經典的演算法將其套用在競賽上，並且加入自己在實作途中對遊戲的理解，進而達到邊時做邊成長的效果。以下將簡單介紹我在這次期末Project中實作的暗棋系統(瑕疵品)，以及在這次期末Project中所學習到、體會到的心得。

System architecture

以下為此次期末Project的class diagram設計，雖然因為時間的和設計缺陷的因素最終版本和這張圖有點出入，但主要架構沒有太大的改變。

- 主要架構中每個**CDCNode**node下hold一張Board，**Board**下面hold35個**Piece** (32顆子+暗棋+空棋+界外棋)以及一個MoveListGenerator，**MoveListGenerator**會根據目前的盤面切換state，不同的State提供不同的走步排序策略，**RuleTable**提供一些吃子、移動子的合法表以及每顆子的基本子力設定，ZobristHashTable提供同形表供**NegaScoutController（不在圖上，由ShiroAI use）**在進行運算時使用，最後還有利用Stack製作一個**NodePool(不在圖上)**儲存回收的CDCNode



Implementation

- **NegaScout Algorithm**
  - 在本次Project中我選擇嘗試實作搭配ID的NegaScout的版本，主要實作在**NegaScoutController**中。
  - 當Main收到server端genmove的指令後就回在更新現在的state之後呼叫 **NegaScoutController::iterativeDeepening(CDCNode*, time)**

```
if(buffer[1]=='r')
{
    myBoard->ply = 0;
    myBoard->rootPly = 0;
}
else if(buffer[1]=='b')
{
    myBoard->ply = 1;
    myBoard->rootPly = 1;
}

myBoard->moveListGenerator->handle(myBoard);
setScoreStrategyByCurrentBoard(myBoard);

//negaScout
if(NodePool::empty()) root = new CDCNode(myBoard, 1);
else
{
    root = NodePool::pop();
    root->copy(myBoard, 1);
}
result = NegaScoutController::iterativeDeepening(root,nextTime/1000);
```

- **Transposition Table with Zobrist hash**
  - 這部分比較沒有什麼特別的設計，最後決定是以ULL為key，總共2^32個index
- **Move ordering**
  - 本次嘗試實作以自己玩暗棋經驗所設計出來的Move ordering，並以State Machine來在每一個genmove的開頭決定此次NegaScout的move ordering (呼叫 **MoveListGenerator::handle(Board*)**)。
  - 主要的設計有5個 **State EarlyGame**，**BothWithKing**，**EnemyWithoutKing**，**WithoutKing**，**BothWithoutKing** 每個State的走步排續策略如下：
    - **EarlyGame** : 吃子 > rule base 翻炮可以吃的位置 > 士旁邊的位置 > 安全的位置 直到自己死兩子or自己有6子以上在場上後切換State (其他state的切換由class名稱就可以了解條件，因此在此不贅述)
    - **BothWithKing**：以吃子為優先，又以兵炮仕為重，在吃的順序中間安插仕帥炮的move判斷是否有機會走路搞死敵人

      ```
      const char BothWithKing::moveOrder[7] = {UG, UK, UC, UP, UM, UR, UN};
      const char BothWithKing::eatOrder[7] = {UP, UC, UG, UK, UM, UR, UN};
      const char BothWithKing::mixOrder[14] = {1,1,1,0,1,0,0,1,1,1,0,0,0,0}; // 0 = move | 1 = eat
      ```

    - **EnemyWithoutKing**：將兵的順序向後排，王和仕的優先度向前

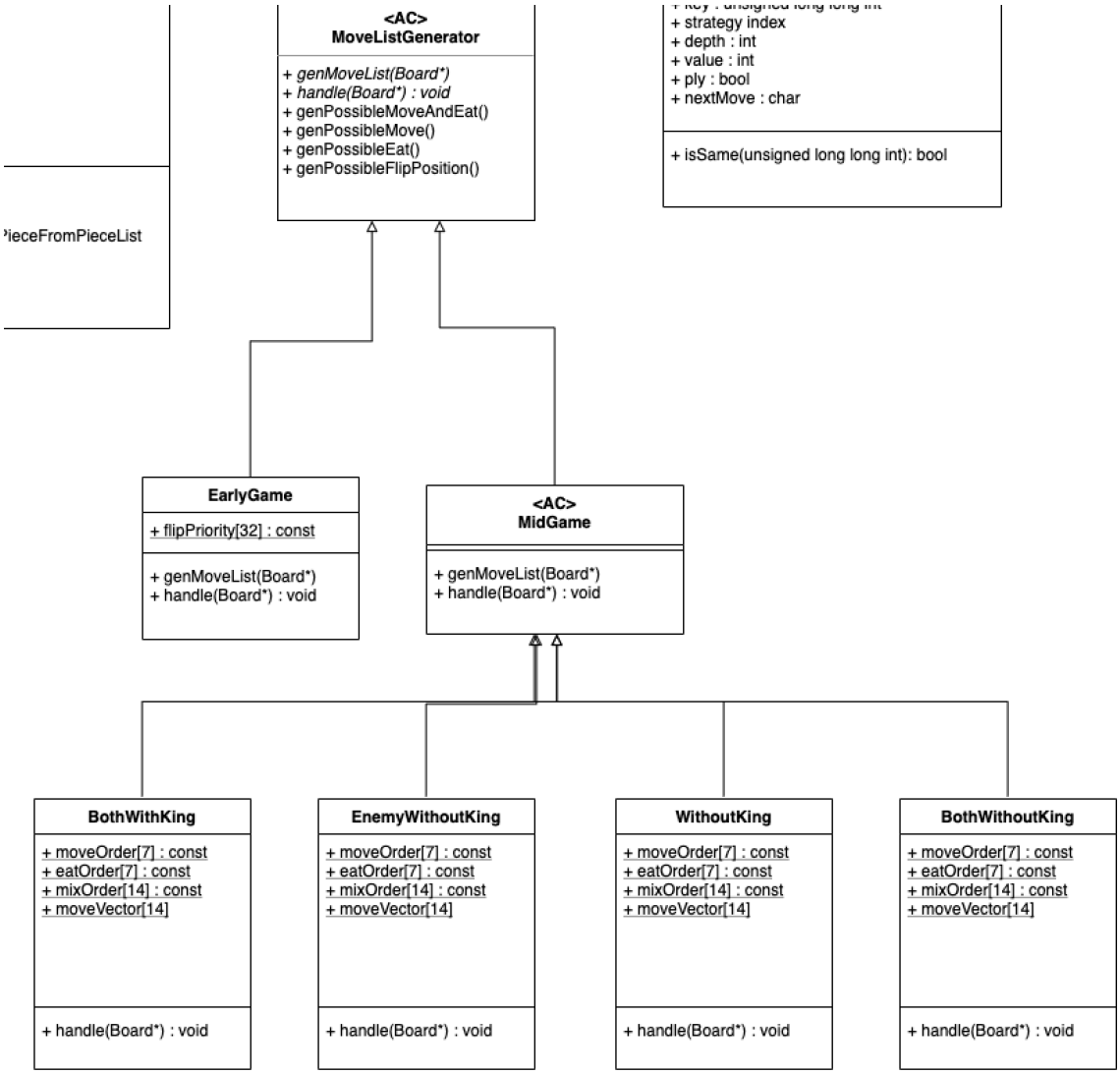
      ```
      const char EnemyWithoutKing::moveOrder[7] = {UK, UG, UC, UM, UR, UN, UP};
      const char EnemyWithoutKing::eatOrder[7] = {UG, UK, UC, UM, UR, UN, UP};
      const char EnemyWithoutKing::mixOrder[14] = {1,0,1,1,0,0,1,1,1,1,0,0,0,0}; // 0 = move | 1 = eat
      ```

    - **WithoutKing**：自己的王死了，因此兵和炮的走步，吃子很重要

      ```
      const char WithoutKing::moveOrder[7] = {UP, UC, UG, UM, UR, UN, UK};
      const char WithoutKing::eatOrder[7] = {UP, UC, UG, UM, UR, UN, UK};
      const char WithoutKing::mixOrder[14] = {1,1,1,0,0,0,1,1,1,0,0,0,1,0}; // 0 = move | 1 = eat
      ```

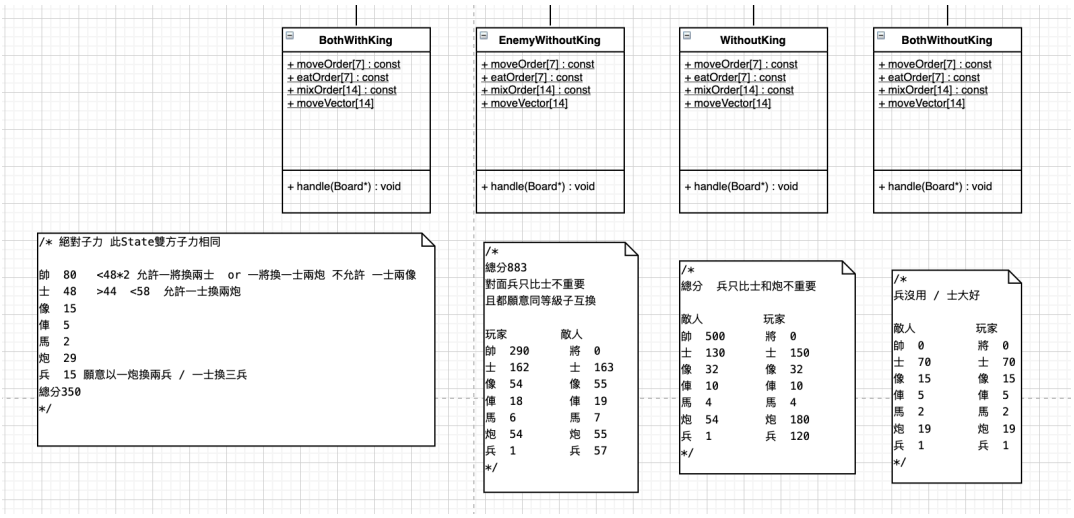
- **BothWithoutKing**：仕最重要，王和卒放到最後

```
const char BothWithoutKing::moveOrder[7] = {UG, UC, UM, UR, UN, UP, UK};
const char BothWithoutKing::eatOrder[7] = {UG, UC, UM, UR, UN, UP, UK};
const char BothWithoutKing::mixOrder[14] = {1,1,0,0,1,1,1,0,0,0,1,1,0,0}; // 0 = move | 1 = eat
```



- ## Chance Node
  - 在ID的第一層有考慮是否翻棋的ChanceNode，但因為時間的關係沒有實作第一層以外的ChanceNode，是否翻棋的分數除了ChanceNode的結果分數外還會根據未翻子的回合數慢慢加權上升
- ## 動態子力價值(審局函數)
  - 基本子力
    - 除了比賽第一局在每個State有不同的基本子力外，最後每局都以固定的基本子力

      (我覺得根據State變換基本子力是ok的但是可能還要有更精準的設計)



    - 動態子力
        - 除了基本子力外每個子的子力還會根據**可以吃的子數**進行加減
        - 此外還進行了淨子差的加減分以及position的分數(越中間越高分)
    - 寧靜盤面
        - 當進行NegaScuot且n==0時會判斷下一步是否會有吃子步，如果有則向下展開一層(只展開吃子步)直到沒有吃子步。
- **CDCNode & Node Pool**
    - 在進行NegaScout時會產生許多CDCNode，如果一直new和delete會很花時間，因此製作了一個NodePool在ID結束後回收不使用的Node
- **RuleTable (for move / jump / eat / score)**
    - 主要的功能如上圖所示在此就不贅述
- **Pondering(未使用)**

  接口實作於ShiroAI.cpp，但是因為時間因素最後沒有將其接上，在測試Pondering時有發生的問題是java關閉時會只關掉client的程式但是不會關閉client fork出的程式，因此fork出來的AI程式要自己定期check ppid來確定自己是否變成殭屍。

What are useful

- Position的動態子力在終局時的效果顯著
- EarlyGame的翻子策略高度的影響到開局的好壞
- NodePool可以非常有效的降低資源回收時間
- 更換State似乎沒有很明顯的效果，可能在吃子步優先的前提下在進行吃子/走步的ordering會更好
- 審局函數非常非常重要！

What are not useful

- 更改每個State的基本子力似乎沒有很大的效果(花很多時間設計的說QQ)

比賽結果

運氣很好的成為了第三名，但是我覺得我寫得不是很好，其中有幾場是帶著嚴重bug上場被打爆的。

總場數	大分	小分	名次
9	??	??	3

Future work

- Refactor 系統架構，因為系統設計時的思考缺漏現在系統的dependency非常高，可以的話希望可以refactor成比較低dependency的版本。
- 目前EndGame時只利用Position的分數來慢慢的接近敵人，未來可以新增EndGame State來實作RuleBase的方式或是蒙地卡羅進行EndGame

結語

感謝老師和助教這一學期的指導，在每次的作業都受益良多，尤其是hw2和期末project都有非常多的收穫，雖然花的時間非常非常多，但是做的都很開心也很值得！

未來會推薦學弟妹修老師的課，希望老師可以繼續開更多精彩的課程，如果有進階課程的話也希望未來有機會可以再次聽到老師的授課！