Name: Kwan Ting Fung ID: 1155110979 Assignment 2 Report

//Sorry for late as I was sick these few days //not COVID, don't worry

Task 2:

1. Advantages of generic code.

```
if self.board.form_mill(x, player):
    print('You form a mill!')
    #remove
    ### TODO
    player.next_remove(self.opponent(player))
    self.board.print_board()
```

The above "player" is the usage of dynamic typing, which it doesn't specify the type of player, it could be human or computer. It is good for function reusage. This is also the advantage of mixed type collection data structures.

2. Adv of "no need to type in extra indice"

```
# Winning condition 2
# check whether the opponent cannot move
can_move = False
for i in range(len(self.state)):
    if self.state[i] == opponent.get_symbol():
        piece_can_move = False
        for j, k in self.edges:
            ### TODO (check every edge to check
            whether there is a legal move)
            if j == i:
                if self.state[k] == '.':
                    piece_can_move = True
```

In the above code, "for j, k in self.edges", j k are the edges value[x,y] respectively in the list, it becomes more simpler which we do not need to initialize indices. Higher writability.

3. Disadvantage at compile time

```
# Start the game
self.board.print_board()
end = False
while not end:
    player = self.next_player()

    if self.num_play <= 12:
        # Phase 1
        x = player.next_put()
    else:
        # Phase 2
        x = player.next_move()
    self.board.print_board()
```

Type checks are need in compile time as each time the variable could have different types. With "while" loop, it checks even more.

4. Disadv of mixed type
   Sometimes we could have x = 100, and in the next line we have x = "adrian", Personally I think it is kind of disadvantage as it is hard to check type bugs. Therefore, in my code in task 1, I don't use mixed type most of the time.

Task 3&4:

Scenario 1:

```python
if self.map.checkMove(newRow, newColumn):
    occupiedObject = self.map.getOccupiedObject
    (newRow,newColumn)
    if type(occupiedObject) == Monster or type
    (occupiedObject) ==Spring:
        occupiedObject.actionOnSoldier(self.soldier)
```

(Python)

I don't need to specify which class "actionOnSoldier" function belongs to, thanks to duck typing and dynamic programming. It has higher writability and more simple, saving little bit of time, as different classes have the same behavior, I just need to call that behavior.

```java
if (this.map.checkMove(newRow, newColumn)) {
  Object occupiedObject = this.map.getOccupiedObject
  (newRow, newColumn);

  if (occupiedObject instanceof Task4Monster) {
    ((Task4Monster)occupiedObject).actionOnSoldier(this.
    soldier);
  } else if (occupiedObject instanceof Spring) {
    ((Spring)occupiedObject).actionOnSoldier(this.soldier)
    ;
  } else if (occupiedObject instanceof Merchant) {
    ((Merchant)occupiedObject).actionOnSoldier(this.
    soldier);
```

(Java)

If it is using Java, it hurts my eye, and I need to specify it is quaking from a duck, or quaking from a chicken.

Scenario 2:

```python
class Map():

    def __init__(self):
        self.cells=[[Cell() for i in range(7)] for j in range(7)
        ]
```

(Python)

```java
public class Map {
  private Cell[][] cells;

  public Map() {
    this.cells = new Cell[7][7];
    for (int i = 0; i < 7; i++) {
      for (int j = 0; j < 7; j++) {
        this.cells[i][j] = new Cell();
      }
    }
  }
```

(Java)

Personally speaking, I think Python has a better initialization of variables in regards of writability and readability.

Python doesn't need to state public private protected, which I don't need to worry too much beyond classes.

And it doesn't need to state the attributes and give memory space (new) again like Java.