Meili Gupta
Anjali Gupta
CPSC 327 Project 6

Design Pattern Write-Up:
We used four patterns in this project: command, factory, strategy, and memento.

We used the command pattern to implement the functionality of moving a piece in the game. The command pattern encapsulated the data and action for Move. Move was an object that took in the board, player number, piece number and direction for moving and building. We implemented execute() and undo() functions in the class. This was very useful because we could call execute() from our main run loop on a Move object. We needed only to return a Move object each turn. undo() was during testing different moves for the Heuristic player.

We used the factory pattern to implement the construction of our three different types of Players: Human, Random, and Heuristic. We implemented a class playerFactory which abstracted the construction of the different types of Player objects. The playerFactory implements one method build_player. In our code, we first call build_player from the initialization of our CLI and pass in the player type as an argument.

We used the strategy pattern to implement the functionality of three different move strategies for each of the Players. Each Player subclass had to implement the choose_move method, and each implemented the choose_move method differently. The benefit of the strategy pattern is that we could choose the implementation of our move algorithm at runtime at each turn based on the Player type.

We used the memento pattern to implement the undo/redo functionality of the game. A memento pattern is set up in the Originator, Memento, and Caretaker classes. We created Memento and Caretaker classes in memento.py. The memento class holds an instance of a game state and has a get_state function which returns the state for the originator to use in restoration. The caretaker class is similar to the example shown in class. However, because our game has redo functionality, we added a pointer to keep track of which turn our current game state is at. This is in contrast to the old method of popping game states off the list when undo is called. In undo and redo, the pointer is respectively decremented and incremented by one. We also added a wipe method, which pops all turns to the right of the pointer when "next" is called.

Individual Statements:
Anjali:
I had a great experience working on this assignment. I greatly enjoyed building the game and working with Meili. As my twin sister, we have worked on projects together before and have experience pair programming. Thus, working on this project flowed pretty smoothly, although we did have some initial issues with GitHub and several merge conflicts when we worked on the same file.

Meili Gupta
Anjali Gupta
CPSC 327 Project 6

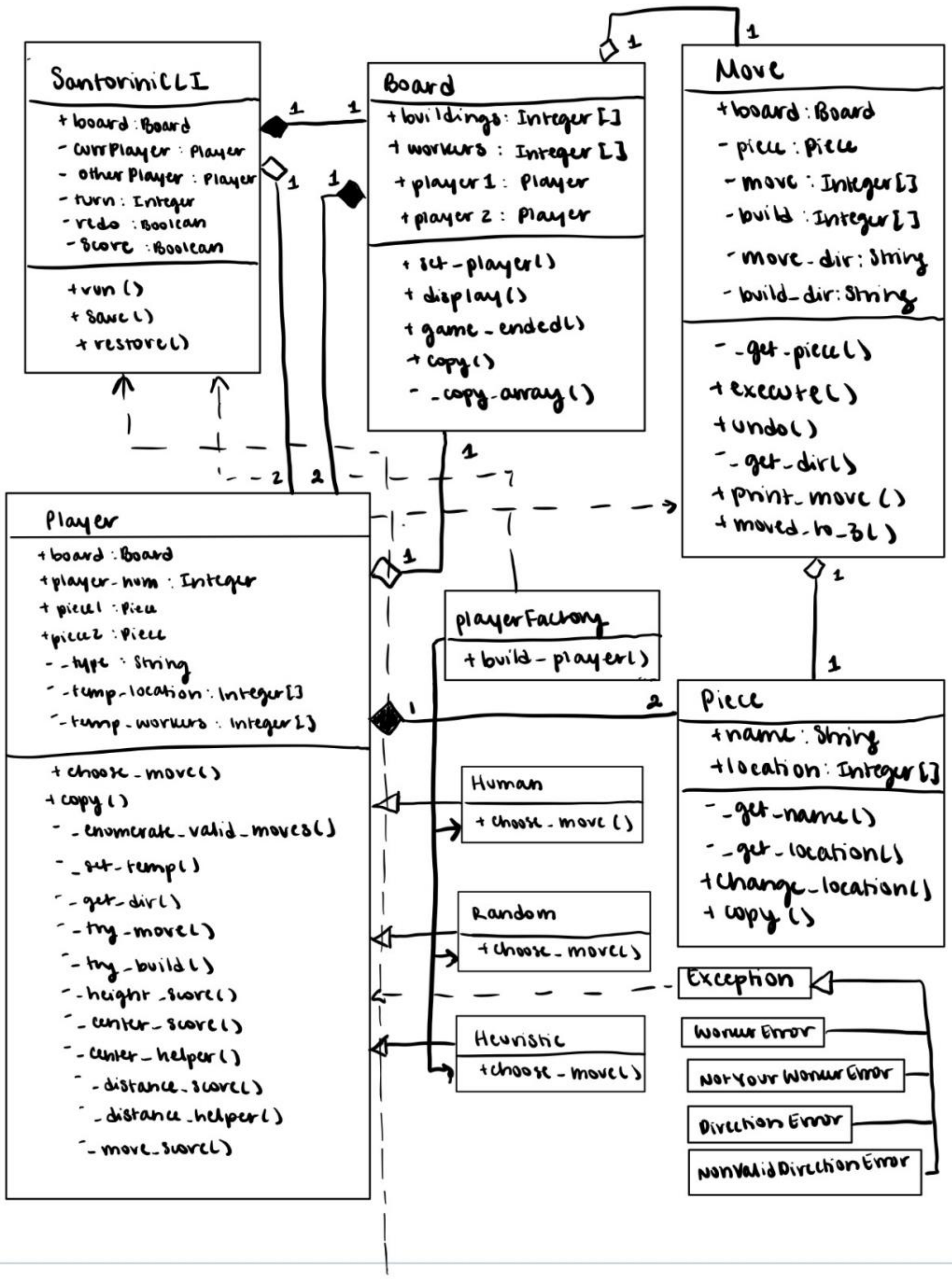
Working with Meili also taught me patience. When I was unsure of how to set up the Originator class in Memento, I learned how to more patiently and carefully explain my dilemma to my partner so she could better help me. I also learned the value of how talking through a bug can make it much easier to solve. In explaining and working through a bug with Meili, it helped me clarify my own thought process and fix bugs faster.

Meili:
I had a great time working with Anjali on this project. We divided the work very evenly, with Anjali working on main.py, memento.py, and the Heuristic player while I worked on all the other files. I was initially very confused about whether I wanted to implement a Piece class, how to organize the relationships between the classes, and how I wanted to store information that I would need in multiple classes, such as the piece locations. Setting up the classes was the most difficult and foundational part of the assignment to me. Drawing multiple versions of UML diagrams was very beneficial.

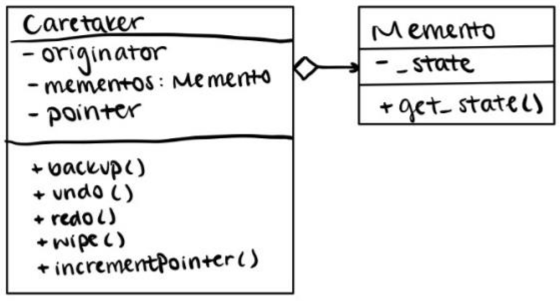
I found myself to be more productive when working alongside somebody else; her presence was motivational. Working together was especially useful when debugging the code. We verbalized our thought processes to each other and would take turns coding up solutions to our bugs. I also found that an extra set of eyes was very useful for catching any mistakes I was writing up in my code.

I’m very glad to have also gained more experience using GitHub, a powerful tool which I am sure to use in the future.

Meili Gupta
Anjali Gupta
CPSC 327 Project 6



SantoriniCLI
+ board : Board
- currPlayer : Player
- otherPlayer : Player
- turn : Integer
- redo : Boolean
- score : Boolean
+ run ()
+ save ()
+ restore()
Board
+ buildings : Integer [ ]
+ workers : Integer [ ]
+ player1 : Player
+ player2 : Player
+ set_player()
+ display()
+ game_ended()
+ copy()
- _copy_array()
Move
+ board : Board
- piece : Piece
- move : Integer[ ]
- build : Integer[ ]
- move_dir : String
- build_dir : String
- _get_piece()
+ execute()
+ undo()
- _get_dir()
+ print_move ()
+ moved_to_3()
Player
+ board : Board
+ player_num : Integer
+ piece1 : Piece
+ piece2 : Piece
- _type : String
- _temp_location : Integer[ ]
- _temp_workers : Integer[ ]
+ choose_move()
+ copy()
- _enumerate_valid_moves()
- _set_temp()
- _get_dir()
- _try_move()
- _try_build()
- _height_score()
- _center_score()
- _center_helper()
- _distance_score()
- _distance_helper()
- _move_score()
playerFactory
+ build_player()
Piece
+ name : String
+ location : Integer[ ]
- _get_name()
- _get_location()
+ change_location()
+ copy()
Human
+ choose_move ()
Random
+ choose_move()
Heuristic
+ choose_move()
Exception
WorkerError
NotYourWorkerError
DirectionError
NonValidDirectionError
1
1
1
1
1
1
2
2
1
1
1
1
2
1



Caretaker
- originator
- mementos : Memento
- pointer
+ backup()
+ undo ()
+ redo()
+ wipe()
+ incrementPointer()
Memento
- _state
+ get_state()