Sukesh Ram & Cole Kaufmann

CS 327

Timothy Barron

12/13/2021

Project 6 Write-Up

Brief Write-Up:

For our project, we used four main design patterns. The patterns we used were: Factory pattern, Template Pattern, Command Pattern, and Memento Pattern.

The Factory Pattern was used for the Factory class. It creates an object without exposing the creation logic to the client and refers to the newly created object using a common interface. We use the Factory pattern to generate the players. Because there are three different types of players, human, heuristic, and random, the internal logic of deciding which type of object to create is handled by the factory class.

The Template Pattern was used for the Player class. This is because for the player class, we have a lot of recycled code. For example, for each type of player, we need the same default implementations of the getCurrentScore() method and the hasMoves() method. However, the different types of players differed in the ways in which they would choose to select the move to take. Thus, the selectMove() method was created as a template method, with divergent implementations in HumanPlayer, Heuristic Player, and RandomPlayer classes.

The Command Pattern was used for the Move class. This is because for the Move class, we have several different types of moves that the workers can take. The point of the command pattern is to essentially have a single execute method that takes no arguments. Upon instantiation of an object, the Move Class does the computational aspects of translating the cardinal direction parameter into the relevant movement on the board of the selected worker.

The Memento Pattern was used for the History class. It essentially allows the user to undo and redo the selected moves that they make. In this case, the Caretaker class is essentially the Manager class which knows the "when" and "why" to capture the originator's state. The Memento is essentially the BoardState class, which acts as a snapshot of the originator's state. The Originator here functions as the History class.

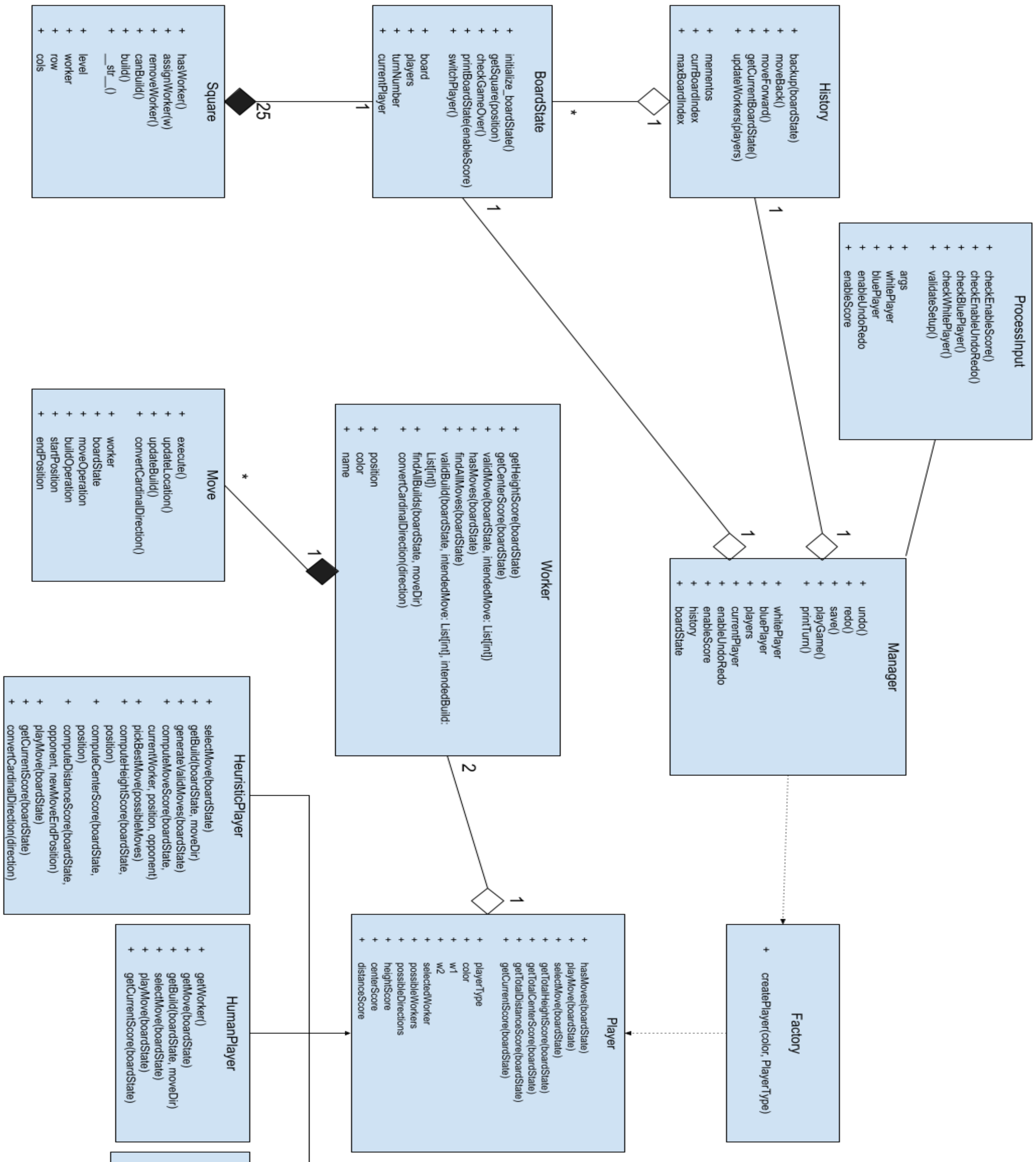
Sukesh's Writeup:

I really enjoyed this project. It was challenging, but incredibly rewarding to work through. I learned a lot of useful skills through doing this project. The first thing I learned was how to break up a large project into manageable chunks. Additionally, I learned how to implement and use various design patterns in order to make my code more efficient, readable, and modular. I also feel as though I became more proficient working with GitHub as a tool. I think also, I became a lot more comfortable working with Python over the course of this project. Through extensive research through the documentation, I was able to learn about a lot of unique pythonic tricks and functions (i.e. .sort() and copy.deepcopy() function).

Cole's Writeup:

Working through this project was a very fun process that taught me a lot of relevant real world skills. I feel as though learning how to work with someone on a programming project is a useful skill. Although it was initially difficult learning to adjust to the fact that Sukesh and I have different coding styles, I think this project was a cool way to learn how to best operate with other people. The project also gave me the chance to practically apply the knowledge we learned in class regarding different design patterns, especially the Memento pattern for the History Class. I also learned how to use GitHub in order to optimize collaboration as I had never used this tool before.

<u>UML</u>:


Square
+ hasWorker()
+ assignWorker(w)
+ removeWorker()
+ canBuild()
+ build()
+ __str__()
+ level
+ worker
+ row
+ cols
25
1
BoardState
+ initialize_boardState()
+ getSquare(position)
+ checkGameOver()
+ printBoardState(enableScore)
+ switchPlayer()
+ board
+ players
+ turnNumber
+ currentPlayer
*
1
History
+ backup(boardState)
+ moveBack()
+ moveForward()
+ getCurrentBoardState()
+ updateWorkers(players)
+ mementos
+ currBoardIndex
+ maxBoardIndex
1
1
ProcessInput
+ checkEnableScore()
+ checkEnableUndoRedo()
+ checkBluePlayer()
+ checkWhitePlayer()
+ validateSetup()
+ args
+ whitePlayer
+ bluePlayer
+ enableUndoRedo
+ enableScore
Move
+ execute()
+ updateLocation()
+ updateBuild()
+ convertCardinalDirection()
+ worker
+ boardState
+ moveOperation
+ buildOperation
+ startPosition
+ endPosition
*
1
Worker
+ getHeightScore(boardState)
+ getCenterScore(boardState)
+ validMove(boardState, intendedMove: List[int])
+ hasMoves(boardState)
+ findAllMoves(boardState)
+ validBuild(boardState, intendedMove: List[int], intendedBuild: List[int])
+ findAllBuilds(boardState, moveDir)
+ convertCardinalDirection(direction)
+ position
+ color
+ name
1
1
Manager
+ undo()
+ redo()
+ save()
+ playGame()
+ printTurn()
+ whitePlayer
+ bluePlayer
+ players
+ currentPlayer
+ enableUndoRedo
+ enableScore
+ history
+ boardState
2
1
HeuristicPlayer
+ selectMove(boardState)
+ getBuild(boardState, moveDir)
+ generateValidMoves(boardState)
+ computeMoveScore(boardState, currentWorker, position, opponent)
+ pickBestMove(possibleMoves)
+ computeHeightScore(boardState, position)
+ computeCenterScore(boardState, position)
+ computeDistanceScore(boardState, opponent, newMoveEndPosition)
+ playMove(boardState)
+ getCurrentScore(boardState)
+ convertCardinalDirection(direction)
HumanPlayer
+ getWorker()
+ getMove(boardState)
+ getBuild(boardState, moveDir)
+ selectMove(boardState)
+ playMove(boardState)
+ getCurrentScore(boardState)
Player
+ hasMoves(boardState)
+ playMove(boardState)
+ selectMove(boardState)
+ getTotalHeightScore(boardState)
+ getTotalCenterScore(boardState)
+ getTotalDistanceScore(boardState)
+ getCurrentScore(boardState)
+ playerType
+ color
+ w1
+ w2
+ selectedWorker
+ possibleWorkers
+ possibleDirections
+ heightScore
+ centerScore
+ distanceScore
Factory
+ createPlayer(color, PlayerType)