To: Mattias ^2
From: Manager
Subject:

Part 1: The pieces of software that are involved

The game Santorini consists of two players, a game board, and pieces for the players (two "workers" apiece and building parts). This game will be implemented in software, with there being a central server managing the board and turns, and autonomous clients that send turn requests.

The client (players) should be able to interface with the game server to determine the state of the board, place workers, move workers, and build buildings. The players should also have one or more strategies of where to place/move workers or build buildings depending on the state of the board.

The game software, on the other hand, is made up of many interconnecting pieces. The server side will need a main program with a few components. There will need to be a board component, which can keep track of all tiles on the board, and know which tiles are occupied by a worker or building and which player they're associated with. Second, on the server there will need to be a turn dispatch component, which receives the turn requests from the clients and sends them to be executed by the board component, then returning the updated board to the client. The game software should also be able to enforce the rules of the game and determine when a game is complete and who the winner of the game is.

In terms of "who" knows "what", there are a couple points to consider. The server will need to know everything concerning the game, including the board state, course of the game, whether a turn request is valid, and whether a player has won or lost. The client will need to know when their turn is, what the game board looks like after a turn request, if the requested turn was valid, if the game is over and the winner of the game.

As common knowledge, there will need to be a common data language such as JSON for all of these different pieces to communicate. In the case of the client, they can send a JSON object consisting of a x, y for worker or building placement, and worker action with respect to buildings. The server, in response, can send a JSON array consisting of an ASCII representation of the board if the move was valid, or send back an exception if the move was not valid for the board.

Part 2: How we are implementing these pieces

For implementation, this will be done in Python 3.6 in the following fashion.

For the server side, there will be a main Game class that has internal attributes consisting of the Board, and a list of connected players. This Game class will manage Players connections and receive Player turn requests, which it will then pass onto the Board. The Game will also need a class method to check the validity of a turn request (if it can be passed onto the Board) and a class method to return the current state of the board to the Player. Lastly, the Game will need a class method to check the state of the Board at every turn and determine if there is a winner, and in that case end the game.

The Board class will contain all of items needed to represent and update the board during the course of the game. It will have a class attribute that is the current state of the board, represented as a 2D array. It will also have two Player class attributes, to keep track of players in the game. The Board will need a class method to print out the current state of the game in an ASCII representation. It will also need a class method to place a Player's piece on the board.

There will be two classes for pieces, a Worker and a Building. A Worker will know what floor it is on and where it is on the board. The Worker will need class methods to be able to move to a certain position, build a building, and move up and down floors in a building. The Building will need a class attribute to know how many floors it has, and a class attribute that contains an instance of a Worker if there is one inside the building. All Buildings will start out with zero floors and each cell in the Board will contain a Building.

The Player class will contain the logic needed to interface with the Game and send turn requests. It will need a class method to send turn requests to the Game. It will also need a class method to print the received Board state from the Game. Each automated player will need to implement the methods specified by the Player abstract class in order to interface with the Game.