To: Matthias^2

From: Matthias

Subject: Design for making a Santorini server and client

Two players must be able to play a game with each other. A player is a client that makes a TCP connection to the server and sends messages based for the decisions they want to take in the game to the server. One piece in the client handle the TCP communication between it and the server. Another piece within the client handles the messages between it and the server and handles user IO accordingly.

There will also be a server that handles running the game and connect playing togethers. One piece of the server handles incoming and outcoming messages over TCP between all the clients. The server also has a queueing system which handles new players who are connected and players who are not currently in a game. It has a matchmaker that pairs up players and puts them in a game together. The server should also handle multiple games between pairs of players which will require making a component for handling games.

The server game handler acts as a single point of truth by storing all information related to the game, the board (buildings and pieces) and turn order. After a game is started, the server sends the board state out to both players. After clients receive the board state, the turn player decides a move and sends it to the server. After each move the server can reject a move if it is invalid or if it is not their turn, otherwise it will resolve the move, inform both players of the result of the move, and the next turn player must decide a move. When the server finds that a move sent wins the game, or a player cannot move, the server sends the final state to both players, notifies them that the game has ended, and informs them of the result. There is a timeout for the turn player handled by the message handler, and if the player hasn't made a move within 3 minutes, the server informs both players that the game has ended, and the turn player immediately loses.

The queue must keep track of which players are waiting for a game and which matches have already occurred as players have to be matched up in tournaments where in which every player gets matched up against every other player.

Internal server parts communicate with each other using internal language structures and clients will talk to the server over TCP using JSON.

The client and server layers first should be implemented using JSON message handling over TCP. The server starts a game with the first two players that connect using clients that interact with players or player programs over command line. The game handler should store a game state that includes a 6x6 array of heights, and 2 pieces per player which have a position value. The game starts by picking the first player randomly who chooses the initial position, and the players alternate choosing initial positions. The rest of the game is played by alternating turns between players until an end condition is reached. On each turn a player must choose a worker to move to any adjacent square (including diagonal moves) and a floor to build on a neighboring building (neighboring building doesn't have to be specified if moving to a square with a building on the third floor). A player can only move to a square if there isn't already a worker on that square and if the building on that square is at most 1 more floor higher than the one the current worker is on. If a player gives the server an invalid move, then the server should respond with an error stating such and wait for another input.

The game ends when any worker has reached a building on the third floor (the player wins, if a player can't move a worker to a two-story (or shorter) building (that player loses), or if a player can move a worker, but not add a floor to a building after the move (that player loses). These conditions should be checked after every turn, and the server should let the players know accordingly if the game ends and who won.

The next thing to be built is the matchmaking system of the server, which should allow tournament matches on the server and abstract the previous section to allow multiple games. When starting a tournament all players should connect to the server, and the server should respond with a new list of players every time another joins. The players should all send a ready message when they want to start the tournament. A list of all pairs of players should then be generated and stored in the matchmaking system, and the server should then attempt to fulfill all matchups. Everyone should be notified when all matches have been played and the tournament is over.