**DESIGN:**

Player Class:

variables:

int[] resources (see note at bottom)

int roadcount

int citycount

int settlecount

String name username

int victory points

int army size

int longest road

boolean isActive

methods:

rollDie()

setters and getters

Resources will be an int array with set indices for each resources:

0 - Wheat

1 - Sheep

2 - Brick

3 - Ore

4 - Wood

5 - Desert (for initial robber placement)

Ports:

0 - Wheat

1 - Sheep

2 - Brick

3 - Ore

4 - Wood

5 - “3-1”

6 - NO PORT

Numbers (for resource distribution):

0 - Water/Desert

Referee:

variables:

int road

int army

boolean gameOver

Player active

methods:

endTurn()

calculateVP()

startTurn()

rollDie()

distributeResources() -- infinite resources

trade(Player p1, Player p2, int[] p1r, int[] p2r)

endGame()

tutorial referee extends referee

networked tutorial referee extends tutorial referee

Game Class -- one instance per player, JFrame, top-level for clients:

variables:

list of Players

Board board

build menu

trade menu

devcard display

methods:

updateBoard(Board b)

updatePlayers(list of Players)

Server Class:

variables:

Board masterBoard

Referee r

list of players

list of clients

Chatbox cb

methods:

updateClient()

startGame()

cleanup()

Chatbox Class:

variables:

jtextfield

jtextarea

jbutton (send)

methods:

sendMessage()

Client Class:

variables:

Game g

Chatbox cb

Player p

boolean isActive

methods:

updateGame(Board b, list of players)

updateChat(Chatbox cb)

Board Class

variables:

Tiles[10][10] tiles

Node[22][22] nodes

Edge[22][22] edges

methods:

Edge[][] getEdges()

Node[][] getNodes()

Tiles[][] getTiles()

Tile class extends Component:

variables:

public final int number (for resource distribution)

public final resource

6 references to nodes (array)

boolean hasRobber

methods:

Tboolean is Water()

boolean hasRobber()

Node[] getNodes()

Node getNode(int index)

void setRobber(boolean b)

Edge class extends Component:

variables:

2 references to start/end nodes (array)

boolean isRoad

reference to owner (Player class)

methods:

boolean isRoad()

Player getOwner()

Node[] getNodes()

Node getNode(int index)

void setOwner(Player p) -- also handles setting isRoad and disables listener

Node class extends Component:

variables:

3 references to adjacent tiles (array)

3 references to adjacent edges (array)

int victoryPoints

boolean isOwned

reference to owner (Player class)

port type

methods:

boolean isOwned()

void setOwner(Player p)

Player getOwner()

Tile[] getTiles()

Tile getTile(int index)

Edge[] getEdges()

Edge getEdge(int index)

void setVP(int pts)

int getVP()

Port getPort()

void addTile(Tile t)

void addEdge(Edge e)

Devcard Interface

execute();

Knight class extends component implements Devcard:

Monopoly class extends component implements Devcard:

Roadbuilder class extends component implements Devcard:

YearOfPlenty class extends component implements Devcard:

Devcard Factory

App Class -- Highest level, instantiates menus to launch game

Settings Class:

variables:

3 jradiobuttons (window size)

jtextfield (default port)

jbutton (submit default port)

jtogglebutton (help messages on/off)

jbutton (go back)

methods:

associated listeners

MainMenu Class:

variables:

jframe

jpanel

5 buttons:

host

join

settings

profile

quit

methods:

associated listeners

Host menu:

variables:

jtextfield (port)

jtogglebutton (# of players -- 3 or 4)

jbutton (submit)

methods:

associated listeners

Join Menu:

variables:

jtextfield (host ip)

jtextfield(port number)

jbutton (submit)

jbutton (go back)

Profile Class:

variables:

jtextfield (nickname entry)

jbutton (nickname entry)

jtextarea (stats)

jbutton (go back to main menu)

methods:

associated listeners

Trade menu:

variables:

two rows of dropdown menus (from 0 to max quantity player\_ has)

submit button

methods:

associated listeners

Build menu:

variables:

4 buttons (grayed if Player can’t afford)

methods:

associated listeners

Devcard menu:

variables:

list of devcards

methods:

associated listeners

**TESTING:**

-- board construction

can we create random board patterns that match requirements? (does our layout algorithm work as it should?)

can we access specific nodes/edges and see their values? (the basics of building)

can we click specific nodes/edges and change their values? (the basics of building)

-- trading

accept a trade in one client and evaluate if the pop-up closes in the other clients and evaluate whether trade was successful (by checking resource arrays)

decline a trade in one client and evaluate if the pop-up does not close in the other clients

offer a trade that one client cannot accept and verify that they cannot accept it

-- making moves (trade, devcard, build, end turn)

given a master board of a known state and a known move, verify that the referee applies the move to the board correctly

verify that the referee only applies valid moves, and informs the player of invalid moves

-- building

client-side: only let player build things that they have enough resources for

build road, make sure that values for longest road are updated correctly

if build road results in new longest road holder, ensure that vps are removed from former owner and rewarded to new owner in server and all clients

-- devcard

“play knight”, ensure that their global (in server, and all clients) army is increased and ensure their global devcard count decreases (ditto for victory points)

“play knight” that gives largest army, ensure that vps are removed from former owner and rewarded to new owner in server and all clients

-- chat box

can the chat receive normal, ‘chat’-type messages from players?

can the chat receive ‘chat’-type messages intended for just one player (PM)?

can the chat send all types of messages to all players?

-- networking

create an object with specific fields on one client, send it to another, and assert that the fields have not changed

update an object in the server and verify that it changes in the clients

ensure that if a client exits or a connection is lost, game ends appropriately

-- robber

ensure that if 7 rolled or knight played, robber is moved and random card drawn from players with settlements on new robber tile

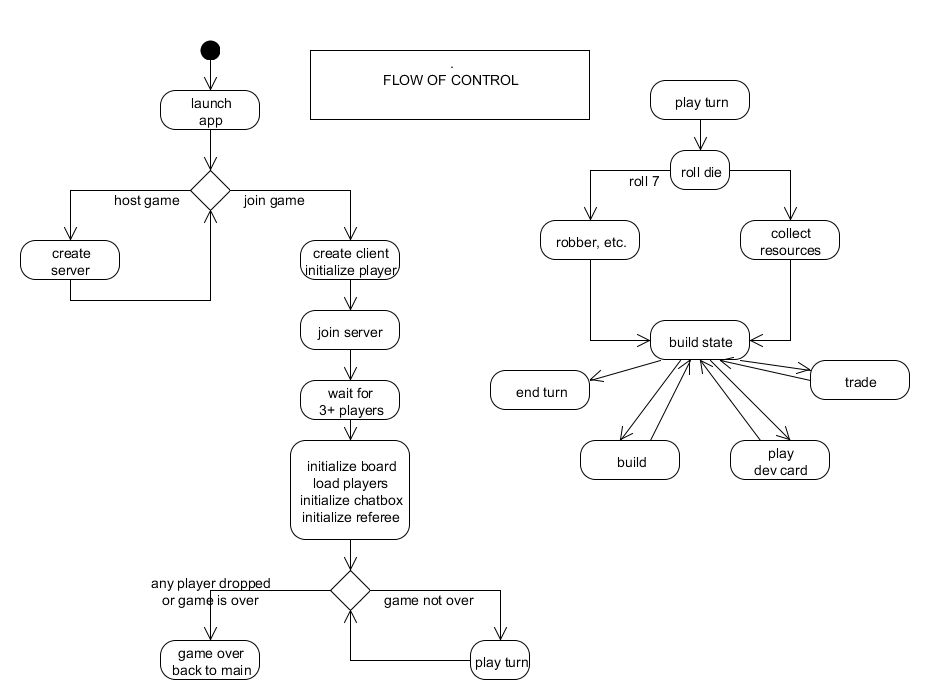
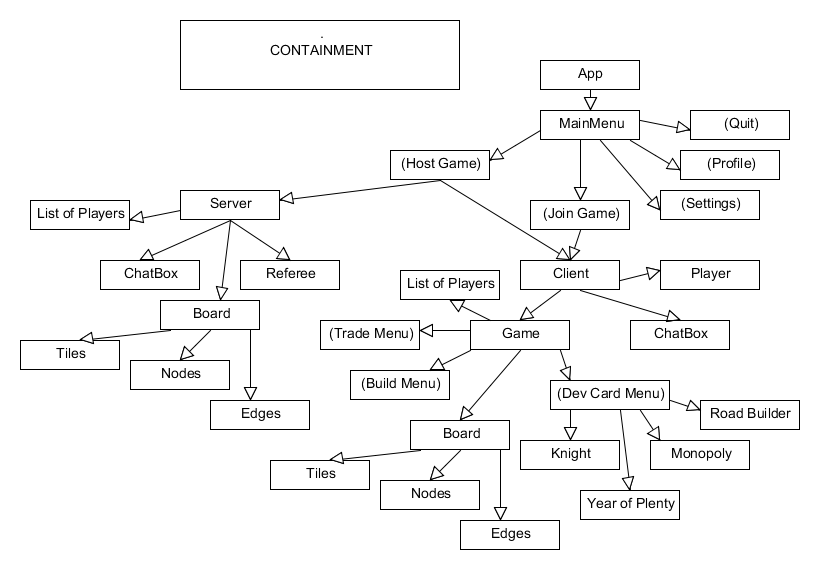
-- resource distribution

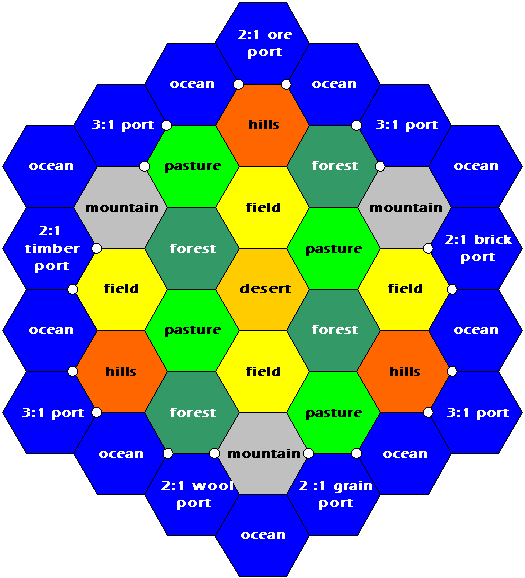
ensure that when dice are rolled, resources are distributed appropriately for the roll

-- end game:

ensure that when a player hits 10 vps, game over pop-up appears until user closes, at which point user is returned to menu screen

----------------------------------------------------------------------------------------------------------------------------





<http://stackoverflow.com/questions/5040295/data-structure-for-settlers-of-catan-map>

Coding Milestone for Settlers of Catan

**Basic Idea**

We will be implementing an electronic, networked version of Settlers of Catan using the standard rules. Players should be able to connect to each other, talk to each other through a chat system, and trade with each other. Furthermore, the board will be “interactive” in the sense that when a player is trying to build or move the robber, they can click the map to make the change. A game will be completed when a player reaches 10 victory points or if one of the users drops out (lost connection). No matter how the game ends, the game should take the player back to one of the menu screens.

**Division of Labor**

While we expect all members to be working on all of the parts (particularly the GUI and game board), we have decided to break the project into the following parts:

* Networking - Setting up a host server and allowing multiple clients to connect to it to play the game. The server will communicate with its clients through a specialized protocol and through the chat box. The server will also send updates that the human clients can read through the chat box (Alex)
* Game Logic - This will check the validity of a given move based on a players resources and board positioning. (Thomas)
* Game board and GUI - Menus for settings and hosting game and such. In game menus for building and trading. Visually displaying the board

**Milestone 1 rubric**

* Set up server that can accept multiple clients at a time (Alex)
* Communication between client and server (Alex)
* All game logic except corner cases (Thomas)
  + internal board representation
  + server and client referees
  + move checking
* Mock-ups of menu screens (Sam/Eric)
  + Main
  + Join
  + Host
  + Profile
  + Settings
  + Placement of swing components
  + Empty listeners
* Mock-ups of assorted pop-ups (Sam/Eric)
  + Trade
  + Build
  + Monopoly
  + Year of Plenty
  + Placement of swing components
  + Empty listeners
* Mock-up of Tile (Sam/Eric)
* Mock-up of Edge (Sam/Eric)
* Mock-up of Node (Sam/Eric)