Dwarf Fortress Clone (DwarfCraft)

Proof-of-Concept Game Project

"Simplified city-building game inspiring by Dwarf Fortress and Minecraft"

General Overview:

- Target Platform & OS: PC / Mac & Windows, OSX, Linux
- Target Language: C/C++
- Graphics library: OpenGL + Magi3 + GLUI 2 (rewritten; not yet public)
- Graphical output: Isometric 3D graphics (polygons, not sprites), voxel world, multi-leveled (i.e. you can move the camera down "into" the earth)
- Game description: Dwarf Fortress clone for the average gamer; multi-player enabled
- Business model: Free to download, not open-source, donation based

Story / Premise

User starts with a set of three dwarfs and must survive natural obstacles and grow a new colony - the ultimate goal is undefined, much like Minecraft, but survival pushes the user to continue colony growth and complexity. To help enthrall the user, colonies tend not to be "self sufficient" in the long run, as new dwarfs are constantly added (dwarf reproduction or immigration) to challenge the user to grow. No population limit exists, as world events are introduced when the simulation sees an opportunity to "prune" the user's growth.

Dwarfs are very unique and act as individually as possible: they are given a job, which they then attempt to complete a subset of tasks within that job-type that the user has defined, as well as "life requirements" such as rest, food, happiness, etc. Much like Dwarf Fortress, each dwarf will have a rolling history of events they were involved with. Unlike RTS games, dwarfs are never controlled individually or by a group, they simply have a priority queue of tasks that the user introduces to the world.

Over time, users will have to create vastly complex cave systems to support their population: farms will have to be built, rivers are to be redirected for power, massive military campaigns are to take place so that the dwarf can get their hand on items to increase their happiness.

Much like Dwarf Fortress and Minecraft, there is a vast amount of content to explore, but the content is spread through breadth, not depth: though there are no end goals, there are countless ways a colony could be built, evolved, and grow to success!

Game Description

Sandbox 3rd person view Sim-like task-manager; open-ended adventure / builder game. Emphasis on simple to play, but can lead to complex user generated content. Mechanisms exist, much like Minecraft / Dwarf Fortress, and so do partial physical and circuit simulations such as flora growth, water spread, mechanism circuitry, etc.

Simulated world includes many environments, biomes, natural geography, cave systems, abandoned cities, but the world itself is limited to help computation. Players simply generate regions, not entire worlds. World is themes high-fantasy (Tolkien-esque world) with elements of steampunk (allows for mechanisms, traps, etc..).

Users indirectly control a colony of dwarfs and can assign world-level tasks; the user never directly controls a dwarf (i.e. no “hero class”, commonly found in many RTS games) nor does group management such as in Starcraft (with the rare exception of military / combat management discussed later). The main interaction / commands are done through “designations” much like Dwarf Fortress. The user selects a region (through more advanced selection than just “rectangle area”; shape-specific selection tools will exist, for both 2D and 3D) and then designates the area to a task - designations cannot overlap but can be merged to create more complex shapes (i.e. an “L” shape comes from two rectangular designations but can be merged if same). Designations range from rubbish, cemetery, collect wood, mine, farm, etc. Designations, as well as materials, are discussed in the Content section.

Crafting, based on job assignment, exists much more like the crafting system found in Dwarf Fortress and not Minecraft’s pattern / recipe based system. Dwarfs are assigned one of three jobs (though secondary jobs exist), and then attempt to do what work is assigned for their Job. Work is defined through either the above mentioned designations or through a crafting table’s work order. Dwarfs, acting independently, will attempt to choose what they see as most important and work in that order. All possible crafting is known from the start, though the real challenge is crafting quality items that don’t degrade / break over time. Similar to the tool-usage system in Minecraft, dwarfs can craft variable-quality items based on the crafter’s level and the used material. Low-quality items break much more quickly that higher-quality items over time.

This game, much like Minecraft and Dwarf Fortress, will have a pyramid scheme of accessible in-game craftables. Before being able to make a certain level of an item, previous items of a lower level must be built. This pyramid, much like Dwarf Fortresses’ jobs / work flowchart, is meant as a way to entice users the build more complex colonies and to reward players that explore and build up inventory of game items.

There are no “secondary worlds” like Minecraft’s nether and soon-to-be-released aether; to keep the world interesting to the user, as mentioned above, procedurally generated historical sites (i.e. abandoned mines), temples with high-value items, and more are inserted into the world. Worlds are generated with histories, that explain the origins of certain structures and world features to the user, increasing the depth of the lore.

Much like the game series “Black and White”, the user should care much about the happiness of the colony as a whole, but the user should not be severely punished if the happiness factor is ignored. This game should never allow the player to “lose” (i.e. all dwarfs are killed) on their own unless through extreme negligence (i.e. famine) or war.

There are world events that are randomly generated based on the game’s time (i.e. date) and colony size. These events include trade caravans (once every season; used to access rare goods), earthquakes (breaks some structures randomly to force the player to refresh his/her building), invasions (like dwarf fortress siege mode), infections (users will have to segregate populations to prevent outbreaks), flooding (tide of rivers go up and may flood entire fortresses), theft (random animals or mobs come in a steal items), and more. These are meant to both challenge the user and, as mentioned, “prune” the user’s work so that older structures are removed and forced construction must take place. This is also to throttle the user’s growth and success, so that the game has a reasonable length of enjoyment.

Like Minecraft and Dwarf Fortress, this game will have day/night cycles as well as seasons. The game universe is based on an 4 month cycle (1 month is equivalent to one season, 4 seasons total), 10 days per month, each “day cycle” (i.e. from morning to night to morning again) being 10 real-world minutes (6 morning and 4 night). This is to pace the game over time. Remember: users can still speed up or slow down up to 4x. In total an entire game year is 4 * 10 * 10 minutes, so 400 minutes in total or 6 ⅔ hours. If the user were to play the game at 4x, this reduces down to 100 minutes per year, and thus 1 ⅔ hours of real time.

Graphics

The target output graphics should follow a sprite-based style that is isometric (not to be confused with 2.5D,commonly found in Final Fantasy games) based, storing data as a voxel engine. This video is a good example of the target graphical output, though the art-style is very much different than the targeted high-fantasy theme.

A world a user plays in is actually just a 3D isometric tile volume rendered from back-bottom to top-front. In terms of the memory map, a game world is a series of discrete 8x8x8 cubes, where each layered “plane” of the world is filled with a single level of these cubes. This allows users to view entire floors without being overwhelmed by data, while still being able to see more than just a single flat tile-bed. Surfaces, either above ground or below ground, should never be smoothed using the “pitched tile set” commonly found in older simulation games, for example in the OpenTTD game.

To help users visualize game data, a dwarf has the defined height of 1 unit tall, and can only move in areas of 2 units or higher caves. Dwarfs may not move up one unit, though they can fall down a single unit sans damage, but any higher of a fall will cause damage and possibly death. It is expected the user either places ladders or stairs for Dwarfs to access higher or lower world cubes. Naturally generated and craftable “half tiles” exist to help the user move between major blocks; these half tiles are comparable to Minecraft “half tiles”. Any non-empty tiles above the middle 2 tiles are self to have opacity to help the user see the dwarfs much more clearly.

This is very similar to my target graphics output. Simple 2D textures turned into 3D voxels (point clouds). Please see the “Technology Notes” below, notes 1 & 2, for a technical discussion of the file format, compression, and general data structures.

Graphics Revision 1:

After doing research, there isn't a reasonable way to do a true "isometric voxel" graphical output. Cubes rendered using a point-cloud (i.e. voxel cube) look either too "spaced out" when seen too close or are too jagged when seen too far. Even at the "best" distance, the edges look terrible (notably the corners) and any camera rotation / movement ruins can show "breakage" (space between the points). Though possible with certain special density rules, the performance would be terrible and not realistic. Graphics cards are great at mass triangle rasterization, not point-clouds.

Without the pseudo-3D effect / point-cloud effect, the isometric approach isn't valid. Going pure isometric isn't valid either, as it might look clean, it isn't anything unique. "Stonesense", an isometric visualizer for Dwarf Fortress, does a great job of displaying data without overwhelming the user. This general approach they have is to only show one level at a time, and on that level only show walls and floors. If there is something to be seen if one were to stand on the edge, then that is also rendered, but is only seen from the side. Again, though this approach is nice, it offers nothing unique and is a strictly 2D approach (not fun).

That being said, voxels are still powerful at representing world data, but one might as well just consider the data structure equivalent to a 3D array. One new option is to save / use word data as a 3D array (so again, just voxels) but render this data as polygons using a marching cubes algorithm. This new style is to render a 3D world using polygons using the voxel source data. Instead of 8x8x8 cubes, entire "layers" of the world will be purely 2D (width x length) and contain two bytes per unit cube. This approach allows floors to be the same data-structure as cubes (i.e. same ID integer, but different meta-byte information). Once voxel data is created or updated, the appropriate 3D model changes are made. A proof-of-concept already exists, as this is a common approach. To keep the desired art style consistent, it is recommended the graphical output be targeted now towards a ragnarok-online style: 3D world, 2.5D character sprites. In ragnarok online, all entities were 2D sprites that were drawn from all four isometric sides, and thus when the camera rotated the 3D world, the 2.5D sprite would swap to one of the major 4 sides. Look at any Ragnarok Online youtube video. Source graphics are already common, since many top-down SNES / 8-bit / 16-bit games would have to draw up/down/left/right, as seen in this Zelda example. A higher-pixel quality image would be the doom guy (note that instead of the four angles, there are in total 8 draw, i.e. corner-views).

This new style keeps the isometric / voxel "look", uses existing and proven technologies (mainly voxel to polygon for the world, and billboard sprites), and may be unique enough to get some attention. Good Minecraft-like output example. Another example. Another another example. Hey another example!

Neat 16-bit retro style for graphics.

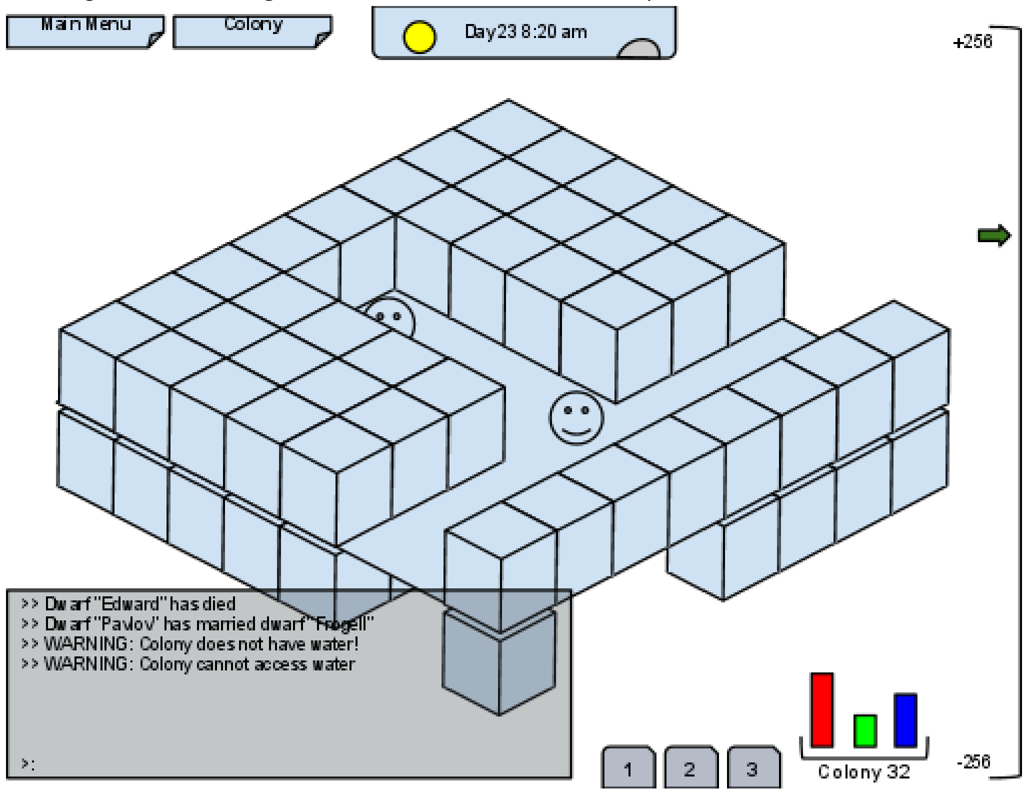
Game Screen

- Main Screen
    - New World

        - User generates a new world based on a seed
        - User then picks camp-site / deployment (setup of dwarfs is optional)
    - Load World
    - Create map
    - Exit

- Game Screen:
  - Overworld view takes full screen
    - Max heights, only see whats been explored (i.e. “fog of war cutoff”)
    - Camera defaults to isometric, left, or right view - players can press WASD to translate across the map, Q and E to up or or down, and R to rotate the camera to the one of three positions...
  - Right side shows a “depth gauge” (vertical)
  - Bottom lists all commands (i.e. “make new designation”)
  - Bottom left is the speed of the simulation (5 speeds + pause, like starcraft)
  - Main menu / configuration is top-left corner (as gear)
  - Messages come out just like in minecraft (bottom left goes up to vertical middle)
    - Colored as needed with events - if click, goes to event
    - Can be scrolled
  - If a craft-bench is selected
    - Shows a list of what can be done, toggle production on or off
    - Set priority factor (sliding scale) (priority is a distribution, so setting “Make tables” to 2.0 and “Make chairs” to 1.0 means that ⅔ will be tables, and ⅓ will be chairs. Stops / grays out upon no resources left
  - If designation is selected
    - Remove designation
    - Change designation
  - Next to designations is a matter of colony health, wealth, and happyness; again similar to sim-city / Black & White games, the higher all three of these, the more productive the colony is and the faster it grows; this is used as a way to help players see what the issue is (i.e. overworking leads to high wealth, but low happiness and low health; an infection leads to low health, etc..)
  - Combat mode is enabled by click a combat button on the bottom-center of the screen next to designations; this is a toggle that the user has to go in and out of between the sim-city controls of designations and the RTS colors
    - Combat is more like an RTS similar to Starcraft: left click to select, along with selection areas and control groups, while right click attacks and target, or area)
  - Dwarf selection:
    - Much like the Minecraft inventory UI: shows name, health, history, status, inventory, current task, etc..
  - Mechanism selection:
    - Simple on/off menu comes up
    - Can be removed or rotated
- Multilayer
  - Put in IP, choose name
  - Note to self: how would I even begin to program this? Server-client model (best option?), or pure client-equality model? I think what might be best is to split the game into a server-client module so that single player is hosted locally on a private port while a multi-player group is the same but with separated controls

- all updates are thus centralized and spit back out to the user (Note from Marshall: possible problem with this is that lag may cause the local user to have great advantages over the connected users).



```
>> Dwarf "Edward" has died
>> Dwarf "Pavlov" has married dwarf "Frogell"
>> WARNING: Colony does not have water!
>> WARNING: Colony cannot access water

>:
```

Sample Game Interface Screen

Game Content

Primary and Secondary Jobs

In total there are three “main” classes with several sub-professions. Every dwarf can individually level each of the three jobs, but only one job may be active at a time. Common tasks that all classes do is general uptake (i.e. remove trash / stray rocks), self update (i.e. eat if needed), and moving / hauling items (i.e. all classes, when something is made or gathered, move it to the appropriate designation; only Miners can place crafted items that are structures such as a workbench).

- Farmer - Supporting class
    - Hunter
    - Gatherer / herbalist
    - Wood cutting
    - Skinning
    - Butchering
    - Farming
- Miner - Building class

    - Breaks rocks / makes paths
    - Repairs broken structures
    - Smooths out stones
    - Places mechanisms
    - Moves and hauls items
- Crafter
    - Cloth Maker
    - Leather worker
    - Armor Maker
    - Weapon Maker
    - Engineer (makes mechanical things)
    - Blacksmith
    - Minting
- Secondary classes - *these have higher priority, but always revert back to the major profession type; each of these are either toggled on or off, but only one sub-type group can be on or off at a time, much like the primary job. Skill quality is directly associated with major level, not job-specific level*
    - Combat classes: ranger, swordsman, dragoon, scout, commander
    - Medic - Heals / hospital attendant
    - Clerk - Does accounting / checks colony inventory

Dwarf Data Structure

The basic dwarf data structure is simple: it is a series of scalars that dictate the status of the dwarf, such as health, age, gender, happiness, etc.

- Active task: what its doing
- Task list: all next tasks
- Job level and experience (for all each of the three job types)
- Overall level (not average of job types; used as a multiplier to all job types)
- Gender: male, female
- Age: dwarfs live up to around ~100 years
- Position: rank in the colony
- Happiness
- Tiredness - how worked the dwarf is, dwarfs should be able to work for 200+ game seconds, then need to sleep 30 seconds
- Story: lists events the dwarf has been involved in
- Is dead: turns into corpse → skeleton over time (once is a skeleton, has no data structure left, is just a block)
- Breathing time (if under water, only has 10 seconds before loosing health; same as minecraft) - can be extended using a diving helm
- Inventory (can hold as much as needed, but only items associated with a task at a time, i.e. if the dwarf is eating)
- Hunger and thirst (two separate factors, diminishes over time based on workload)
- FROM MARSHALL: Relationship interests for dwarf reproduction?

        - Good point: I've added gender, reproduction likelihood, etc.. Would be neat to create a genetic algorithm-type reproduction system, so that successful genes reproduce faster.

Dwarf Logic - "Logic" Flow Chart

All dwarfs follow the below simple logical flow chart. Based on what is near the dwarf, as well as its own status, and current commands the user has issues, it may do one of many different things.

- If medical emergency (dwarf's health, food, water, tiredness, or air are too low)
    - Go to the appropriate designation (i.e. bed, food stock, etc..)
    - If medical emergency not resolved, dwarf dies and turns to a corpse (and eventually a bone block)
- Else if happiness is too low
    - Disobey orders randomly
    - Self-assign new orders randomly
    - If happiness is critically low
        - Go crazy (undefined at this moment, maybe kills self or others?)
- Else if in combat group
    - If no issued commands: Attempt to gear up - randomly select weapon class (ranger, swordsman, dragoon, scout, commander)
    - Else if attacking: attempt to move to any open adjacent square for combat
    - Else if in control group / selected (similar to starcraft): stay near other groups
- Else if assigned to one of three major jobs
    - Look at jobs queue which comes from assigned designations and/or craft-benches
    - Randomly choose a task based on user-assigned priority (mentioned above: though there is a priority queue, users can place weights to tasks and thus have one task be much more factors)
        - Gather needed resources into inventory (i.e. tools or raw materials)
        - Move to work location
            - Movement is based on the below-mentioned movement algorithm
            - If a rail system exists, the dwarf will use it (the rail must be ready and have carts at both ends)
        - If the area is currently locked (i.e. being used), stall for 5 seconds, try again
            - After a certain number of stalls, give up and inform the user a new task for this dwarf is being selected
        - Attempt task until completion (i.e. craft item, cut down tree, slay)

Raw Resources / Materials

All of the following are raw resources that can be collected in the game. Some of these are minerals found in the ground / mines, while others are cultivated or caught. In no particular order:

- Dirt / Sand:
  - Sand (fine, red sand, gritty)
  - Gravel
  - Clay
  - Dirt (May or may not have sand on it)
  - Mud (water logged dirt)
- Rocks:
  - Coal (Most common)
    - Can be used as a source of power, like lava, and water
  - Basalt
  - Mica
  - Bauxite
  - Chalk
  - Sandstone
  - Shale
  - Limestone
  - Granite
  - Graphite
  - Marble
  - Slate
- Ores: *Great example of varying stone types*
  - Copper (Most common, Lowest Quality)
  - Tin
    - Copper + Tin = Bronze
  - Silver
  - Gold
  - Platinum
  - Iron
    - Can be made to steel
  - Titanium
  - Obsidian
  - Adamantine (Most rare, Best Quality)
- Flora:
  - Oak (Best)
  - Maple
  - Beech
  - Pine (Worst)
  - Seeds (Of any flora)
  - Red / Green / Blue Flowers
  - Wheat
  - Hay
  - Mushroom
  - Moss
  - Bush

        - Smaller foods: Roots, Plants, Berries (i.e. grapes), Nuts, figs
        - Larger foods: beets, rutabagas, carrots, celeriac, turnips, apple, orange
    - Vines

- Fauna / Other:
    - Leather
    - Meat
    - Bones
    - Animals (deer, slugs, bats, fish, big fish, dog, moose, monkeys, birds, cows, etc..)
        - Animals cannot be explicitly controlled, but are programmed to "roam" and self replicate as needed
        - Animals, much like dwarfs, can die from lack of food or water
        - Animals can be fenced together but may consume all food in that region if not large enough - can be fed by hay, particularly useful when animals are underground
        - Minecraft's old system is a great example
    - Wool
    - Fire (cannot be collected; but can happen and spread)
- Environmental:
    - Water
    - Lava
    - Oil
    - Fire

Work Benches

The following are structures (single cube or multi-cube) that can be used to accomplish a task such as butchering (the Farmer must take the animal to the building to be butchered) or crafted (dwarf needs raw materials). All benches, excluding the wood and stone work benches, need power. This power can either come from a furnace (i.e. burn materials to power aspects) or from a wire mechanism. This pushes the user to pace their growth and gather needed materials to setup a bigger colony.

- Wood Crafting (carpentry)
- Stone Crafting (stone generation)
- Engineering (builds mechanisms)
- Kitchen (Food processing)
- Forge (powered either by wood, lava, etc..)
- Weapon smith
- Armor smith

Crafted Items

When the user selects a work bench, they are given a list of all possible items they could craft with, along with the list of qualities (based on available resources). Items that cannot be crafted because of a lack of required resources are grayed out to not confused the user. All crafted items not only have a base quality factor depending on the material, but this factor improves based on the crafter's skill and a random (very low) "chance of legendary quality" which produces a very strong, regardless of quality, item.

- Blocks: [Example series]
    - Of any raw sand / dirt / rock
    - Special blocks: glass
- Bars:
    - Of any raw ore; may need to be merged
- Benches:
    - All benches are crated (“First” bench is in the starting colony)
- Power sources: (Powers adjacent locations or wires)
    - Furnace - Uses wood, coal, or anything that can burn
    - Water paddle - Must have a water block above and a water block below; removed water above and pumps it below
    - Magma rod - Place in a pool of magma vertically to power
- Light:
    - Torches (wood + coal)
    - Any furnace makes light
- Tools: (can only hold two at a time)
    - Pick (underground)
    - Axe (felling trees)
    - Shovel (sand / dirt)
    - Hoe (farmning)
    - Skinning knife (skins animals)
    - Butchers knife (butchers animals)
    - Sheers (gets wool)
    - Fishing rod (gets fish)
    - Fishing net (more like a “fish farm”, place once, gather over and over)
- Mechanisms: *Mechanisms bring a fun level of complexity and automation into a player’s world. All mechanisms are graphically designed with input and output lines and tend to be singe-cubes. Fore example the “and block” (a single cube with four sides) takes in two to tree inputs, while outputs only one side. The user, when setting down mechanisms, chooses direction, but must explicitly power.*
    - Wire (needs copper bars)
    - Repeater
    - Switch (either on or off; not a button)
    - Logical blocks:
        - AND block
        - OR block
        - NOT block
    - Vertical / Horizontal pump (pushes liquids from one side to another, user chooses direction)
    - Flood gate (open and closes all directions)
    - Rail (speeds up transportation of dwarfs: for example if a group is mining far away, instead of walking, they choose the minecart to either move themselves)
        - Needs Minecarts
    - Mechanical Doors (not to be confused with regular door)

    - Pressure plate (if pressed, set wire to high)
    - Traps:
        - Spike (out of ground, always up)
        - Automated spike (if powered, moves up to deal damage if walked over)
        - Note that the “pump” and “flood gate” can push in and out lava
        - Note that pressure plate (as noted before) activates upon ANYTHING over it
        - Fall traps (just have mobs fall)
        - Piston wall (only goes up - when powered, moves all the way up to crush or trap)
    - Animal / mob attractor (block that glows and attracts either animals or mobs nearby)
- Storage:
    - Crate (for grouping resources such as ores, food)
    - Barrel (for water)
    - Bucket (used to move things around)
- Furniture
    - Table
    - Chair
    - Cub
    - Bed
    - Casket
- Specialty:
    - Door
    - Fence
    - Diving Helm (adds 60 seconds to a dwarfs diving time)
- Clothing, Weapons, & Armor: (One set for each major type of metal)
    - Sword
    - Shield
    - Bows
    - Arrows
    - Pole-arms
    - Daggers (comes as a pair)
    - Heavy Swords (two handed)
    - Helmet, chest plate, leggings, boots of varying metals
    - Clothing (not armor; this increases happyness) (general items; always just 4 types: head, chest, waist/leggings, feet)
- Processed foods:
    - Hay (dried grass)
    - Flour (milled grain; done it kitchen)
    - Bread (made from flour)
    - Drinks (beer and wine needs kitchen and barrels)

Designations type:

- Rubbish (places all trash in this area)

- Mine (gathers and stockpiles minerals / rock / etc)
- Set (fills volume with stone type)
- Crafted-goods stockpile (tools, empty barrels, etc..)
- Food stockpile (all foods and liquids)
- Stone stockpile (raw rocks / unprocessed)
- Ore stockpile (raw metals / unprocessed)
- Processed ore and stones stockpile (all blocks and metals placed here)
- Farm (area to be farmed, does not store food)
- Wood (i.e. fell trees, gather wood)
- Flood (place water in this area using buckets; self note: users should try channels first)
- Barracks (used for both sleeping and healing, though needs medic assigned person)
- Hall (the default location where they go to eat, socialize, etc)
- Armory (weapons racks; separate from tools)
- Grave (gets rid of bodies)
- Protect (all combat-flagged dwarfs rush to this area if a mob has entered it, regardless if it leaves later on)

Map Generation:

When the user chooses to start a new game, a "world map" is generated using a group of user-defined options similar to Dwarf Fortress and Minecraft. This world map is not the one the player will play in, but is instead a global map the user may choose a subsset of to play in. Subset selection is based on pre-defined sizes based on multiples of 2 (i.e. 128, 512, 1024, etc). They are as follows, based on scalar values, from 0% - 100%:

- World Name (string)
- Seed (optional, just gets hashed)
- Biomes
  - Planes
  - Woods
  - Desert
  - Tundra
  - Savanna
- Weather (more similar to a precipitation map)
  - Precipitation
  - Temperature ranges (i.e. world varies a ton, or only varies around a specific value)
- Water:
  - Rivers
  - Water level
- Other: (boolean)
  - Cave complexity
  - Ruins
  - Temples

    - Dungeons
    - Villages

- Simulation time (The longer, the better / more developed the world is)

A proof of concept image can be found here; this comes from a feature suggestion prototype on r/Minecraft.

Once the user generates a world and then selects a segment, the local world (the one the user plays in) is generated. This is based on several procedures: the first is the world's topography. This is based on a localized perlin noise map. Next comes the precipitation and temperature layers; both are user-defined but dependant on the world selection. The map is then modified based on biome rules: if the given column is both high-elevation, cold temperature, and high precipitation, a snow-capped mountain is generated. A low-elevation, high temperature and low precipitation form deserts. The sizes of these combines areas are then either grown or shunk using standard bleeding algorithms based on the user's preference. The next major step is to introduce rivers, where lake origins are randomly placed at common locations (i.e. less likely in deserts, more likely in forested areas, generally always above the ocean level), then filled to the lakes height, then rivers are generated that always go towards the ocean, and if forced, will create valleys / tunnels (even if this local map doesn't have an ocean, the global maps data is used). Cave systems are placed randomly by the half-line method (i.e. draw a line randomly underground, cut it in half and move the center points of the two new lines a little, repeat, then eventually hollow out). Minerals are placed based on a density map (i.e. when placing gold, choose a random point between a target range, place, and bleed a little if needed to create more gold blocks). Finally, ruins, villages, etc.. are placed randomly ontop of the world at locations that fit their size requirements geographically (i.e. place a village on a flat hill, rather than on top of water).

Over time, in-game weather will only change based on the seasons: deserts will become colder, snow is more common near mountains, etc..

The creators of Dwarf Fortress have a good discussion on the matter of world generation.

System Simulation

When running the game world, there are many independent entities, world physics, and lighting that has to be simulated at run time. A classic approach would be to thread each major element, and let the host's operating system do the thread management. In this game, since there may be hundreds of entities, threading might introduce too much overhead to the host OS. Instead, what the system may do is event-based computing and delayed computer models. The first, event based computing, means a system will only be updated or computed at the last minute (i.e. why pre-compute water flow if the user has yet to open a gate / block?). The second is similar to a time-share system: do nothing while the entity is waiting for a timer to release. This second method is useful because many of the game entities are performing game actions (i.e. eat, mine, etc..) that are just time-based and thus have nothing to computer. Each entity can have an update(dT) function that takes in the amount of time elapsed since the last update. Internally, each entity does a simple state-condition check: If in state X, how much time left? If time left is over, switch to next state, else, stall (i.e. return). It is possible that on certain events, such as a dwarf needs to compute a path, can take longer than a single frame (since updated are assumed once a frame is rendered). Such components should be threaded, so that the main thread can continue on standard computation.

Technical Notes:

1. Though the camera cannot rotate (i.e. isometric view constraint), we can do slight rotations to help the user see things by stretching the height and width distance between voxels.
2. To save, we can use octrees (http://en.wikipedia.org/wiki/Octree) since many areas are similar, then each 8x8x8 block can be saved using a very simple linear-representation encoding (i.e. the string "10x3" means that for the next 10 blocks, they will be of type block 3)
3. All blocks should be two bytes: either one (8 bits) or one and half byte (12 bits) are used to identify the block, while the remaining bits are left as meta-information, such as if a wire is on or off, water height, block direction, etc.
4. Path planning: use a global graph-based system, not a local search discrete elements system. Example: Look at all the entrances and exists of a major cube, then use a macro-level path planner. For example, imagine a "Y" shaped road. Instead of saving each discrete point that makes the path into the path planner, only save four nodes: three as the tips of the "Y" shape, and the middle split node. Then the path planner can much more effectively choose a path. THOUGH there is an issue with this, such as the "?" path, where there are multiple entrances into a node, but only one valid one to the path - this may cause path planning bugs but can helped by creating a "walked path" graphics system - i.e. the more walked on a path is, the more flat / smooth it gets.
5. Water flow algorithm:
    - Water blocks are a byte value (0 - 255), where anything less than 10 evaporates)
    - Water blocks only do global simulations every "game second" (so you see animated flows)
    - A single block of water without any adjacent blocks will evaporate
    - Blocks will flow in all open directions - dividing itself as needed (for example, if a block of water is next to two open)
    - There are source blocks, such as at the edge of maps that always replicate at the same height (i.e. does a full fill-in at all adjacent blocks except above) - if dammed, all that aren't adjacent to the source blocks will start to disappear - opening holes can allow for lower flow (i.e. a converts a river to a stream, or redirects it to a farm, etc..)
6. Survivors of Ragnarok is an interesting game that seems a bit similar...
7. Experiments to do for optimization:
    a. OpenGL:
        i. Immediate mode - using iteration rendering
            1. 128x16 world size: 6.5 FPS
            2. 64x16 world size: 43 FPS
            3. 32x8 world size: 60 FPS
        ii. Immediate mode - using Octree rendering:
            1. 128x16 world size: 21 FPS
            2. 64x16 world size: ~60 FPS

3. 32x8 world size: 60+ FPS

iii. Display list mode - batch the entire thing
   1. 128x16 world size: 22 FPS
   2. 64x16 world size: 60 FPS
   3. 32x8 world size: 60 FPS

iv. Display list mode - per-chunk rendering (EXACT same as batch)
   1. 128x16 world size: 22 FPS
   2. 64x16 world size: 60 FPS
   3. 32x8 world size: 60 FPS

v. VBO mode - entire world as single VBO:
   1. 256x16 world size: ~16 FPS
   2. 128x16 world size: 45 - 60 FPS (60 when *not* moving)
   3. 64x16 world size: 60 FPS
   4. 32x8 world size: 60 FPS

vi. VBO mode - chunke'd world
   1. Note: have different chunk sizes for the geometry and data:
      a. 8-16 ^3 is good for data, as it keeps it all page-friendly for the OS / hardware
      b. 32x32 is a good size for 3D geometry

vii. *Intelligent VBO - face removal*
   1. *Needs to be tested but pretty obvious it'll speed up because of same concept as backface culling*
   2. *Note to self: not that critical as players will only see a flat level at a time anyways...*

b. Data structures:
   i. Use an octree data structure?
      1. Currently testing...
      2. **Complete**: VERY memory effective, though when iterating to get data to draw, you can either do an x,y,z index search (slow) or do a recursive update (slightly dangerous but faster)

8. Terrain generation references:
    a. Twenty Sided article
    b. Minecraft Lighting system
    c. Ruins / Dungeons:
        i. http://donjon.bin.sh/dungeon/about/
9. How mechanism / circuits are simulated:
    - Only do event-based simulation (i.e. don’t simulate every second, simulate on pressure-plate change,
    - On / off states propagate through adjacent tiles for up to a distance of 16; can repeat using a repeater mechanism (this is done through a simple recursive adjacent-fill algorithm)
    - All mechanisms have designated inputs and outputs on each side (small ‘I’ and ‘O’) that make it clear how to order